PARSING WITH A STATISTICAL DEPENDENCY MODEL

Daniel Zeman

PhD thesis
Advisor: doc. RNDr. Jan Hajič, Dr.

Univerzita Karlova v Praze
Matematicko-fyzikální fakulta
Ústav formální a aplikované lingvistiky
Praha, 2004

1 Abstract / Abstrakt

This thesis presents a statistical parser of Czech. We analyze step by step its evolution and the merit of its various parts. The parser is based on the original approach of dependency bigram modeling. Although there are other parsers that have been originally developed for English and their adaptations for Czech perform better than ours, we demonstrate that our parser makes different errors and thus it can help the better parsers to become even better.

Tato disertační práce popisuje statistický parser (syntaktický analyzátor) češtiny. Krok za krokem v ní analyzujeme vývoj parseru a přínos jednotlivých jeho částí. Parser je postaven na metodě přímého statistického modelování závislostí mezi slovy, která je originální i při srovnání se zahraničními pracemi. Přestože se nepodařilo tímto způsobem získat nejúčinnější možný nástroj pro syntaktickou analýzu češtiny a existují české adaptace původně anglických parserů, které si vedou lépe, ukážeme, že díky odlišnému přístupu dělá náš parser chyby jiného druhu a lze tedy jeho kombinací s lepšími parsery dosáhnout výsledku, kterého žádný z nich není sám o sobě schopen.

2 Acknowledgements

I would like to thank Klára for her endless support. Besides taking care of our home and our child, besides her own job and career, she not only had the inexhaustible reserve of smiles and encouragement. She was always willing to help me with anything, and always prepared to the highest sacrifice: being expelled from our home computer ☺.

Thanks to my parents and others who pushed me to put the research results into this form by periodically asking things like "How long shall we wait for your becoming a doctor?"

Thanks to my supervisor, Jan Hajič. From his credits I would like to point out the crucial one: that he brought me to the field and did not let me after my graduation to vanish in an average software company. Thanks to all my colleagues in the Center for Computational Linguistics of the Charles University, for their support and encouragement. And thanks to my friends and mentors abroad, prof. Fred Jelinek of JHU, Baltimore, prof. Aravind Joshi and Anoop Sarkar of UPenn, Philadelphia.

A thank of a different sort goes to all the kind taxpayers who fed my family and me during this research. Of course, there is much more (data, software, hardware…) for which we thank to the following fund sources:

The GAČR grant No. 405/96/K214 took care for the initial specification of PDT. The project of the Ministry of Education of the Czech Republic No. VS96151 (Laboratoř jazykových dat) enabled the annotation of PDT 0.5. A significant part of the Model One was developed at the 1998 Workshop on Language Engineering, CLSP, Johns Hopkins University, Baltimore MD, and supported by the National Science Foundation Grant No. #IIS-9732388. The largest part of the research and its presentation (work with PDT 1.0, investigation of subcategorization, regular expressions) has been supported by the Ministry of Education of the Czech Republic project No. LN00A063 (Center for Computational Linguistics). Finally the experiments with the superparser were funded by the Czech Academy of Sciences program "Information Society", project No. T101470416.

Although the persons and institutions mentioned above helped me a lot, any conclusions and opinions expressed in this material are mine and so is the responsibility for any possible mistakes.

3 Contents

3.1 Index of Figures

4 Introduction

"Parsing is rediscovering the relations between words of a sentence, a procedure that can be fed with a flat string and outputs a tree representation of that string." I used these or similar words when — just a couple of years ago — I described my job to a guy whose research domain was different from mine. It was not different too much: he was involved in a speech recognition project. That is why I was quite surprised as the man replied: "Why do you do that?"

That episode now contributes twice to this Introduction. First, I learned that it is worth motivating the parsing problem in more detail even when assuming that potential readers are more or less related to the natural language processing (NLP) community. And second, by showing how parsing could aid speech recognizers we immediately start the motivation itself.

Classical speech recognizers were based on two main models: an acoustic model ranked words by the probability of their matching given input acoustic signal, and a language model reranked whole *sequences of words* in case a "strange" sequence arose. Typically, the language models were n-gram models based on probabilities that a given n-tuple would occur in text (n = 2 or 3). Such models did not always provide the best hints. Consider the sentence

(1) He is a resident of the U.S. and of the U.K.

Let us now assume that for some reason the acoustic model prefers the reading

*(2) *He is a resident of the U.S. and* that *the U.K.*

A bigram model would support the latter (false) hypothesis. Collins (1999), where this example comes from (p. 5), reports that according to the Wall Street Journal (WSJ) data in the Penn Treebank (Marcus et al. (1993)), the bigram [*and that*] is around 15 times as frequent as [*and of*]. This leads to approximately 10 times greater probability assigned to the incorrect string than the probability assigned to the correct one.

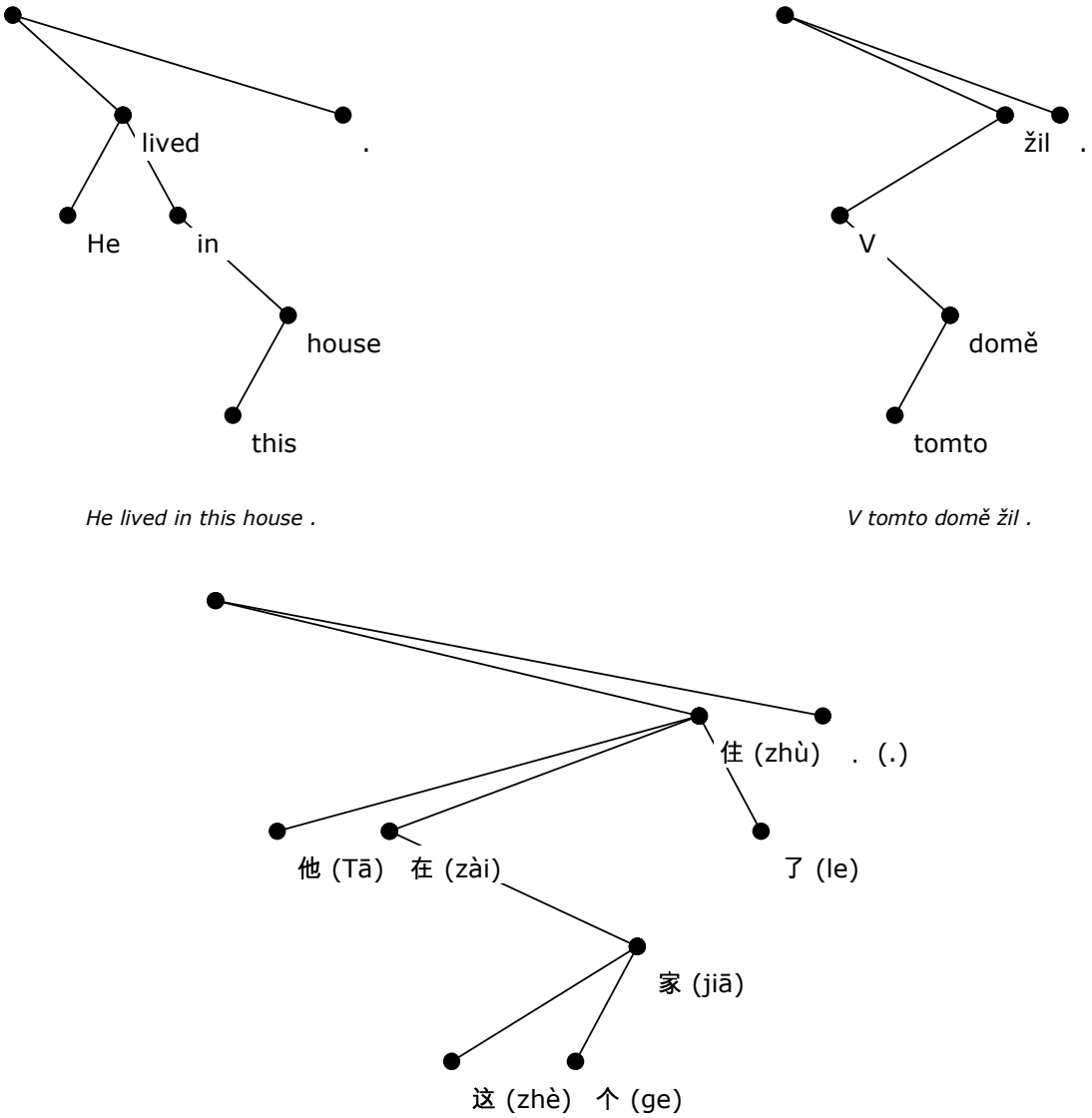
In contrast, one might imagine a more sophisticated, syntax-aware model that would discover the long-distance relation *of — and* and thus the need for a second *of* after the *and*. In fact, such models have recently been invented by the speech recognition community, one good example being the Structured Language Model (Chelba and Jelinek (1998)). An integral part of such model is a syntactic parser. Without advances in parsing, the model would hardly be imaginable.

Another instance of parsing as a useful (if not necessary) preprocessing step is in machine translation. Consider the following English sentence and its Czech and Chinese counterparts:

(3) He lived in this house.

(4) V tomto domě žil. (Lit: In this house he-lived.)

(5) 他在这个家住了。(Tā zài zhè ge jiā zhù le.) (Lit: He/she in this piece-of house live action-completed.)



He lived in this house .

V tomto domě žil .

Tā zài zhè ge jiā zhù le .

Figure 1: Surface structures for the sentence "He lived in this house" in English, Czech, and Chinese.

None of the three languages uses the same word order as the other. In other languages we would find yet other word orders, and there are languages considering more than one word order plausible. Moreover, the *number* of words differs. In *pro-drop languages,* there often is missing the word serving as subject; in languages like Czech there are no articles etc. No matter which language is the source language and which is the target one, a machine-translation (MT) system obviously cannot translate the input sentence on a word-by-word basis. It has to apply some word-order variating procedure. Our claim is that if the input and output of an MT system were structures rather than plain sentences (i.e. strings), the translation itself would be much easier. Then the "word order procedure" would be split into two parts, the first part being integrated in parsing

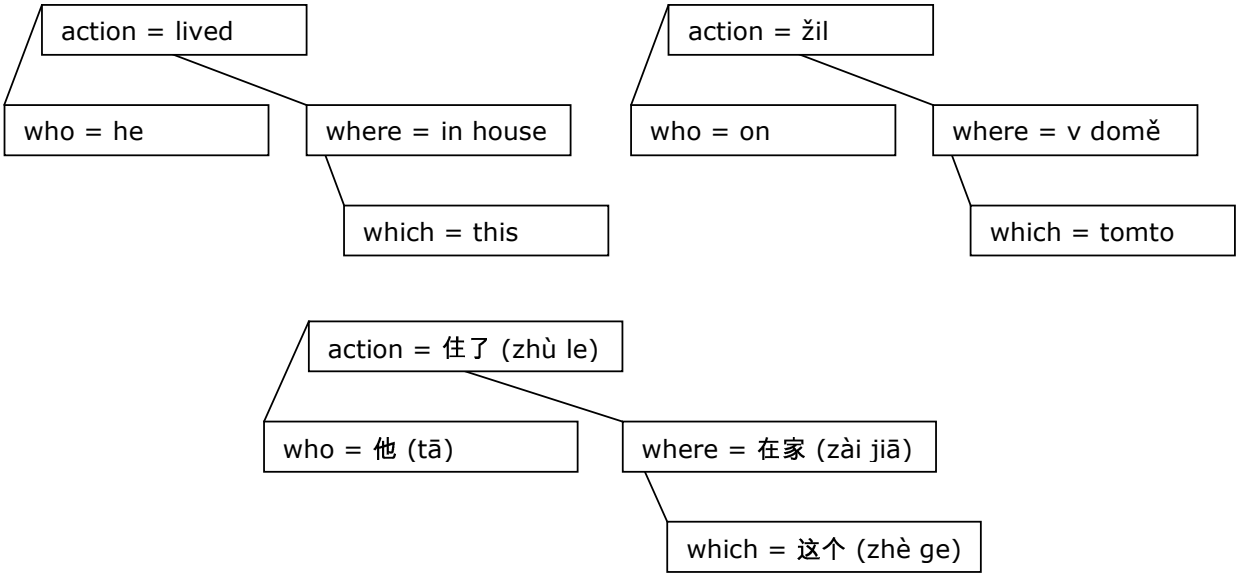


Figure 2: Deep structures for the sentence "He lived in this house" in English, Czech, and Chinese.

(building input structure), while the rest would be generating the output sentence with a plausible word order from the output structure. Such process is illustrated in Figure 1 and Figure 2. Figure 1 shows sentence structures that may have been created by a parser.[1] Figure 2 shows a kind of semantic frames that could be easily constructed from the parser output and would be immediately translatable one to another. Actually the possibilities of such structure-based MT are currently being investigated at the Center for Computational Linguistics (CKL, http://ckl.mff.cuni.cz/) in Prague (Cuřín et al. (2002)).[2]

Parsing can also help word sense disambiguation. For instance, the English verb *to stop* would be translated to Czech as *zastavit* in sentences like "The bus stopped in Times Square" and as *přestat* in sentences like "He stopped talking". Again, the structure output from a parser would tell us what arguments the verb has. We would then be able to select one of the verb's *subcategorization frames*, and thus one of its meanings and one of its translations.

Our final example documenting parsing usefulness is a grammar checker. A grammar checker, in contrast to a spell checker, can point to a word that exists in the language but its particular usage in a given sentence is not plausible. Probably the most

[1] There are many different syntactic theories each of which defines its own sentence structure. We will declare one of the theories as the framework for this thesis later in this Introduction; so will we precisely define *our* syntactic structures. For now, the example structures are provided "as-is" without any formal definition.

[2] The example illustrates usefulness of *deep* syntactic structures in contrast to *surface* ones. However, this thesis investigates only surface parsing. Our assumption is that the road to the deep structures leads over the surface ones. The surface structures represent an intermediate step, simplifying the whole analysis process. We believe that the structural shift from the surface to the deep level will be rather easy; the most challenging part will not be finding the deep *structure* but tagging the dependencies by *functional tags* (labels "action", "who", "where", "which" in Figure 2).

remarkable targets of a grammar checker in many languages are agreement violations, like the number agreement violation in the English sentence (6) and the gender agreement violation in the Czech sentence (7):

(6) **The coffee are very hot.*

(7) **Koťata byly utopeny.* "The kittens/Neut. were/Fem. drowned."

Some parsers are based on a grammar and can directly tell that a sentence has no valid syntactic structure according to that grammar. Some even augment their grammars with special error generating rules enabling to build a structure for some incorrect sentences and to point to the errors in them (cf. Kuboň (2001)). Other parsers use statistical methods and are capable of assigning probability to a structure. When normalized according to the number of words in the sentence, such probability can serve as an indicator of possible grammar errors. Sentences whose probability falls below some predefined threshold can be highlighted as suspicious. As the statistical parsers typically compute the structure probability from probabilities of its components (phrases, word dependencies…), they can be queried on the suspicious words as well.

4.1 Theoretical background

This thesis mainly focuses on parsing of one particular language: Czech. The language itself quite narrows our selection of theories that can serve as the theoretical ground to our formulation of the parsing problem. Better expressed, we need not narrow the selection but there is a strong preference for one theory: the dependency syntax as incorporated in the Functional Generative Description of language (FGP, see Sgall et al. (1986)). There are three reasons for such preference.

Reason 1 is traditional. Since the first half of the 20th century the Czech linguistics is known and appreciated as the so-called Praguian School. The view of sentence syntax as a set of dependencies of one word on another is an idea accompanying the school from its cradle. The constituent-based syntactic structures, introduced by Bloomfield (1933), got their popularity later thanks to Chomsky (1957), and their usage never became so widespread among Czech grammarians as it is the case for English.

Reason 2 is linguistic. Whole books could probably be written about the pros and contras of the dependency vs. other approaches. The scope of this thesis and the qualification of mine do not allow to include them here but at least we can very briefly summarize some points:

1 There is no common consensus about the necessity of having non-terminals in the syntactic structure. Many major theories use the notion of phrases that have names – non-terminals in the terminology of Chomskian grammars. The dependency syntax does not use non-terminals.

2 Even the parsers grounded in constituency-based theories (e.g. Collins (1999)) profit essentially from the knowledge of the head of a phrase, where the head is a word that in some way or another governs every other member of the phrase, and vice-versa, all other members of the phrase depend on or modify the head. In dependency syntax, such relations are expressed directly in the structure.

3 There are constructions allowed by the grammar of the language that cannot be described by syntactic structures of immediate constituents. Later in this thesis we will mention a class of such constructions, the non-projective constructions. Although we will show that they form a problem that is marginal and hard-to-solve, it is fair to choose a theory of enough expressive power to cover them.

Reason 3 is purely practical. As our parser employs statistical methods, we need a corpus to collect statistics on. Currently the only publicly available syntactically annotated corpus for Czech is the Prague Dependency Treebank (PDT, see Hajič (1998)) and if we want to use it, it is natural to adopt its theoretical background as well. From the opposite point of view, the group around PDT may profit from a parser that generates structures in agreement with the PDT specifications.

4.2 Dependency structures

After having motivated the selected framework we proceed to its formal specification. Throughout this thesis we usually will (for simplicity) use the word **word** for what is sometimes called **token:** a word is either a real word, or a number, or a punctuation mark. So not all words in our sense are separated from the others with spaces. More precise rules might be needed for word segmentation of Czech text, especially for multi-character punctuation marks or some forms of numbers. However, our definition of word is sufficient as we define it for the purpose of this thesis, not the parser itself — the parser requires a word-segmented text on input and thus relies on the rules of the word segmentation tool.

A **sentence** is a sequence of words. Each word in a sentence has a unique **index** in the sentence that reflects the word order: the starting word has the index 1, the final word (usually a full stop) has the index n where n is the number of the words in the sentence.

A **dependency** $d(w_i)$ of the word w_i is an integer number from the interval $<0;n>$. The case $d(w_i) = 0$ means that w_i does not depend on any other word in the sentence; a non-zero dependency is equal to the index of the word that governs the word w_i. Whenever it will be clear from context we will make no distinction between numeric value of the dependency and the word it refers to. In other words, we often will denote the **governor** of w_i by $d(w_i)$ rather than $w_{d(wi)}$. Dually the word w_i will be called the **dependent** of $d(w_i)$. We also will use the alternative terms **parent** (for the governor) and **child** (for the dependent). A dependency can be represented graphically as an oriented line connecting two **nodes**, one for the governor and the other for the dependent. The line must be oriented to distinguish the governor from the dependent; we will express the orientation by the Y-coordinates of the nodes in diagrams: the governor will always be placed higher than the dependent. We will not use arrows to express the orientation since there is no consensus whether the arrow should point from the governor to the dependent or vice-versa. The X-coordinate will preserve the word order: if $d(w_i) < i$, then the governor will be displayed more to the left than the dependent; otherwise, it will be to the right.

The word w_i **dominates** the word w_j if either $d(w_j) = i$, or there is a k such that w_i dominates w_k and $d(w_j) = k$.

While we are defining *mathematically* what is a good dependency, we do not put any *linguistic* constraints under which one particular word can depend on the other. Such rules have already been defined by the PDT annotation team and the PDT annotation should fit the rules. So — unless explicitly stated — we will use the PDT as the only authoritative source of judgment whether a dependency is *linguistically* correct or not.

A **dependency structure** *DS* for the sentence $S = w_1,\ldots,w_n$ is the sequence of dependencies $d(w_1),\ldots,d(w_n)$ fulfilling the condition that no word dominates itself (i.e., there are no **cycles**). Note that the condition implies that at least one word in the sentence must be independent ($d(w) = 0$). Further note that while the values in the sequence denote the indices of the governors, the ordering of the sequence denotes the indices of the dependents so the sequence contains all necessary structural information for the sentence. A dependency structure can be represented graphically as a rooted tree with n+1 nodes. The added node is the artificial root of the tree where independent words "depend" on. In accordance to the above, we will place the root as if it was a word with index 0.[3] For all nodes we will keep the above convention for their X- and Y-coordinates. We will use the terms dependency structure, **dependency tree** and **tree** interchangeably. Should we occasionally need a different kind of tree (e.g. a phrasal tree), we will explicitly state that.

Example: "0,1,2,0", "0,0,0,0", and "2,0,4,2" are some valid dependency structures for a four-word sentence (no matter what the words are). There are many more valid structures, namely 125 total, as there are $(n+1)^{n-1}$ valid structures for an n-

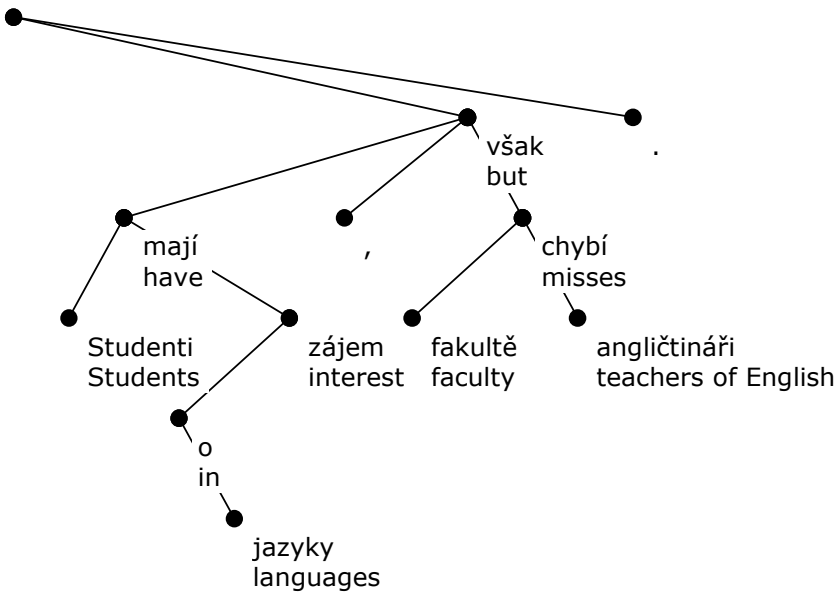


Figure 3: Studenti mají o jazyky zájem, fakultě však chybí angličtináři.
Students are interested in languages but the faculty is missing teachers of English.

[3] Should we occasionally need a "word" for the root, we will use the symbol "#" as does the PDT team. Note however that we don't count this "word" into the number of the words in the sentence.

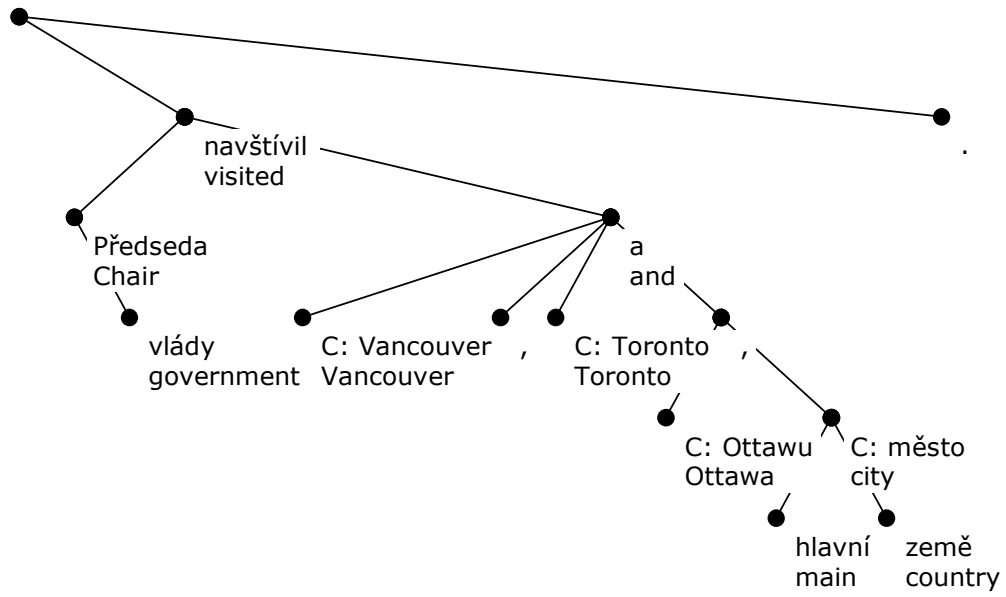


Figure 4: Předseda vlády navštívil Vancouver, Toronto a Ottawu, hlavní město země.
The Prime Minister visited Vancouver, Toronto, and Ottawa, the nation's capital.

word sentence. To give a more linguistic example, consider the sentence (8). It has eleven words and thus over 60 billions of valid trees. However, only one of these is linguistically correct according to PDT: "2,8,5,3,2,8,9,0,8,9,0" (see Figure 3).

(8) Studenti mají o jazyky zájem, fakultě však chybí angličtináři. "The students are interested in languages but the faculty is missing teachers of English."

The fact that for every word of the sentence there is a corresponding node in the tree implies that some dependencies are rather technical, without any linguistic interpretation (cf. the comma in sentence (8)). Besides that, there are dependencies capturing language phenomena of a non-dependency nature. Most non-technical dependencies represent the linguistic relation of **subordination** of one word to another. However, some relations in the sentence are better described as **coordinations** than subordinations. Whole the coordination may govern a word, and the interpretation is the same as if that word simultaneously depended on each member of the coordination. Analogically, coordination can depend on a word, and the interpretation is same as if that word simultaneously governed each member of the coordination. But there is no internal hierarchy among the members themselves: they all rank equally and no one is subordinated to another.

In PDT, coordinations are represented in dependency structures as follows. An auxiliary word (usually a coordinative conjunction or a comma) becomes the governor of the coordination. All coordination members are attached as its dependents, as are the words really subordinated to the coordination. We will soon define s-tags, the syntactical tags. In coordinations, special values of s-tags distinguish between coordination members and the words subordinated to the coordination. An example follows (for the structure see Figure 4).

(9) *Předseda vlády navštívil Vancouver, Toronto a Ottawu, hlavní město země.* "The Prime Minister visited Vancouver, Toronto, and Ottawa, the capital of the country."

To avoid any misunderstanding in future, we should now stress the difference between the terminology some readers may consider usual and the terminology defined for the purpose of this thesis. First: broader sense of coordination. If the sentence (9) was in PDT, the second-level "coordination" ("Ottawa, the capital…") would actually be tagged as apposition, rather than coordination. Both constructions indeed differ linguistically in how they combine meanings but topologically they are similar. We will thus cover both by the term *coordination* whenever we do not state anything else. Second: dependency is not subordination. Subordination is a linguistically motivated, formally unspecified vertical relation between parents and children in the tree, and contrasts with coordination, a horizontal relation between siblings in the tree. Both are represented using dependencies in a dependency structure.

4.3 Layered description of language

FGP adopts the view of the language as a system that can be described in levels. The higher levels are closer to meaning, the lower levels are closer to surface representation (a text, an utterance). Natural language analyzing procedures traverse the levels from bottom upwards and use output of a lower level as the input to a higher level. Language generating procedures traverse the system in the opposite direction, instantiating the meaning represented on higher levels with the means of the lower levels. The number and classification of the levels may differ according to the needs of the application; PDT distinguishes three levels.

The lowest level of PDT is the **morphological level (ML).** It treats a sentence as a sequence and concerns mainly with the structure and ambiguity of words. As input it gets a plain string. The output of ML (or the representation of the sentence on ML) contains the same string enriched by additional information such as word boundaries, sentence boundaries, and the output of **morphological analysis (MM).** The latter assigns to each word all combinations of lemmas and morphological attributes that may have generated the given word form, regardless the actual sentence context. The values of the morphological attributes (as gender, number or case of nouns) are encoded in morphological tags or **m-tags.** The m-tags naturally encode the part-of-speech as well and to some extent they correspond to the POS tags used in English corpora. Note however that the richness of the Czech morphology causes our tagset to contain thousands rather than tens of possible tags.

The m-tags shall not be confused with the s-tags that will be defined on the next level. Throughout this thesis the default interpretation of "tag" is "m-tag".

The morphological annotation on the output of ML may optionally be accompanied by the **morphological disambiguation (MD).** It is typically performed by a **tagger** (machine or human) and its goal is to select for each word the correct lemma and tag with respect to the context. The disambiguated output contains certain percentage of errors.

Example: a sentence and its representation on the morphological level.

(10) Při se, jak chceš. "Quarrel as you want to."

Word	**Lemma**	**M-tag**	**English**
<f>Při	<MMl>pře_^([soudní]_spor)	<MMt>NNFS3-----A---- <MMt>NNFS4-----A---- <MMt>NNFS6-----A----	a lawsuit
	<MMl>při-1	**<MMt>RR--6----------**	by, during, on…
	<MMl>přít_^(se)_(o_něco)	<MMt>Vi-S---2--A---- <MMt>Vi-S---3--A---4	to quarrel
<f>se	<MMl>s-1	<MMt>RV--7----------	with
	<MMl>se_^(zvr._zájmeno/částice)	**<MMt>P7-X4----------**	oneself
<d>,	**<MMl>,**	**<MMt>Z:----------**	,
<f>jak	<MMl>jak-1_;L_^(živočich)	<MMt>NNMS1-----A----	a yak
	<MMl>jak-2	<MMt>J,----------	how
	<MMl>jak-3	**<MMt>Db----------**	how
<f>chceš	**<MMl>chtít**	**<MMt>VB-S---2P-AA---**	to want
<d>.	**<MMl>.**	**<MMt>Z:----------**	.

The above table (excluding the column with English translations) shows SGML-style morphological annotation of the sentence (10), as output by the morphological analysis by Hajič (2004). The lemmas contain some word sense disambiguation (the extensions "*-number*"), sometimes a semantic or stylistic category (like the "_;L" attached to *jak-1*), and sometimes even an explanation comment (the extensions _^*comment*). The m-tags are strings each of 15 characters where the first two characters specify the part-of-speech: in this example there are nouns (tags starting with N), verbs (V), pronouns (P), prepositions (R), conjunctions (J), adverbs (D), and punctuation marks (Z).

Note the high degree of morphological ambiguity in the sentence: except of punctuation and the verb *chceš*, every word allows more than one interpretation (lemma & tag).[4] If we apply a tagger, it will select for each word only one lemma and one tag based on context. The bold entries in our table are the choices made by the on-line tagger at http://nlp.cs.jhu.edu/~hajic/morph.html; although the page does not refer to relevant publications, it is probably a recent version of the tagger of Hajič and Hladká (1998). The annotation in PDT provides output of two taggers, identified as "a" and "b". The tagger "a" is a maximum-entropy-based tagger; see Hajič and Hladká (1998). The tagger "b" is the HMM-based tagger described in Hajič et al. (2001) (but without the rule-based module described there). The annotation of the first word of sentence (10) might then look like the following:

```
<f>Při<MMl>pře_^([soudní]_spor) <MMt>NNFS3-----A----
<MMt>NNFS4-----A---- <MMt>NNFS6-----A---- <MMl>při-1 <MMt>RR--6----------
```

[4] The average number of tags per word in the training part of PDT 1.0 is 2.76.

<MMl>přít_^(se)_(o_něco) <MMt>Vi-S---2--A---- <MMt>Vi-S---3--A---4
<MDl src="a">při-1 <MDt src="a">RR--6---------- <MDl
src="b">přít_^(se)_(o_něco) <MDt src="b">Vi-S---2--A----

The middle level of PDT annotation is called **analytical.** Two important kinds of information are added on this level: the dependency structure and the so-called analytical functions (afuns), or **s-tags** (syntactic tags). We have introduced the dependency structures but we omitted the linguistic guidelines for their construction; the readers interested in the guidelines should refer to Hajič et al. (1999). The s-tags encode types of dependencies (the most important are: subject (Sb), predicate (Pred), object (Obj), adverbial modifier (Adv), attribute (Atr); coordination and apposition roots are tagged as Coord or Apos respectively; the members of coordinations or appositions have _Co/_Ap attached to their s-tags (Sb_Co, e.g.)). Despite the possibility of syntactic ambiguity the current version of PDT assigns only one dependency and s-tag to each word.

The highest level of PDT is the **tectogrammatical level.** It further modifies the dependency structure but there are significant differences from the dependency structures in our sense. Auxiliary words are hidden and get no dependency (not even the 0-dependency). Lexical words are retagged by *functors* or **f-tags**, corresponding to their semantic roles. One may think of the structures in Figure 2 as of a simplified example of what is going on on the tectogrammatical level. We need no more formal definition because for the rest of the thesis this level will be out of our sight.

We will finish this survey of PDT layers by putting our work into their frame. We are building a parser that operates on the analytical level. We always will assume that the **input to the parser** has already passed through the morphological analysis and contains the morphological annotation. Usually we also will profit from the existence of taggers and demand the input to be morphologically disambiguated.

The **output of the parser** will not be a full analytical annotation. The parser will generate dependency structures but not the s-tags. Nevertheless, it will sometimes use the hand-annotated s-tags in PDT for learning.

4.4 The goals and structure of this thesis

The goals of this thesis are twofold. We, of course, want to present a parser that achieves as high accuracy as possible. That is our main goal.

Besides documenting the parser, the other goal is to document the *development* of the parser, including some of the steps that turned out to be less than useful. The main idea we will follow is unique among the different approaches to Czech parsing, and I even cannot name a non-Czech parser that would be comparable to ours to a significant extent. That is why it is important to document the dead ends as well, so they are prevented from being considered again when the parser is further developed in future.

The structure of the thesis is subordinated to this second goal. In Chapters 7 and 8 we describe two generations and many subversions of the parser. Chapters 9 through 12 discuss some issues more or less related to the main topic of parsing.

Brief evaluation of each parser modification is presented immediately and summarized in a table at the end of the actual chapter. The *standard evaluation method* is defined in Chapter 5, while Chapter 6 describes the baseline to compare the results to. Various supplemental evaluations are presented in Chapter 13. Improving the state of the art is done in Chapter 14 by combining several different Czech parsers. Finally we survey the related research and compare other approaches to ours (Chapter 15), and conclude (Chapter 16).

5 Standard evaluation method

As NLP will never produce 100% correct analyses, it is important to be able to say how well a particular tool performs. Various evaluation metrics used in various NLP tasks usually share one common feature: most of them tell us the percentage of success of solving the task on independent **test data.**

Sometimes (for instance in machine translation) it is difficult to tell what is a success. For dependency parsing, there are several possible metrics we may use. We will discuss them in more detail in Chapter 13 where we also give results of those metrics for our parser and our test data.

Before the final evaluation, throughout the thesis we will need a measure to compare the contribution of two or more modifications to the parser. When not explicitly stated otherwise, in all these comparisons we use a single metric, called **accuracy** (or dependency accuracy, as opposed to sentence accuracy). The (dependency) accuracy is defined as follows.

Let us have a test sentence *S* with words $w_1 \ldots w_n$, and a reference tree for that sentence *RT(S)*. The reference tree has been built by a human. Let us denote the tree generated by the parser as *GT(S)*. For each word w_i, there is a reference dependency $rd(w_i)$ in *RT(S)*, and a generated dependency $gd(w_i)$ in *GT(S)*. Each dependency is represented by an integer number — the index of the parent node in *S*. We say that $gd(w_i)$ is correct whenever $gd(w_i) = rd(w_i)$. The **accuracy** of the generated tree *GT(S)* or simply the accuracy of (analyzing) the sentence *S* is the number of correct dependencies divided by the number *n* of words in *S*:

$$A(GT(S)) = \frac{\sum_{i=1}^{n} (gd(w_i) = rd(w_i))}{n}$$

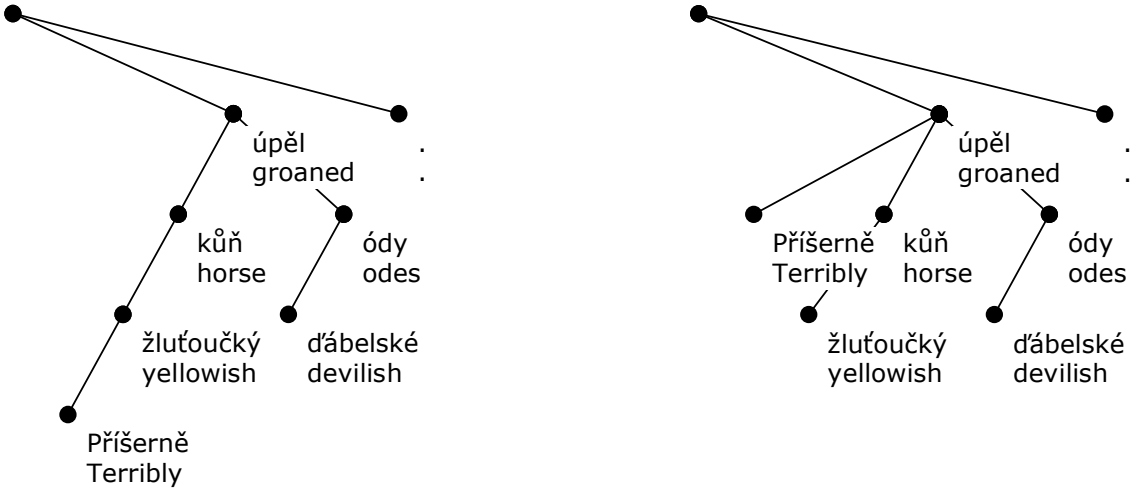


Figure 5: Příšerně žluťoučký kůň úpěl ďábelské ódy.
A terribly yellowish horse groaned devilish odes.

To give an example, consider the sentence *Příšerně žluťoučký kůň úpěl ďábelské ódy.* ("A terribly yellow horse groaned devilish odes.")[5] Look at Figure 5. The left tree shows the reference tree for that sentence while the right one may have been generated by some parser.

The generated tree differs from the reference tree in one dependency, out of seven. This gives us the accuracy A = 6/7 = 86%.

Similarly the accuracy of a set of sentences is the number of correct dependencies in the trees generated for the set, divided by total number of words in the set. Number of sentences with zero incorrect dependencies does not matter. The accuracy (or performance) of a parser is the accuracy of trees generated by the parser for a given test set.

5.1 Prague Dependency Treebank

If not stated otherwise, throughout this thesis we will use various versions of the Prague Dependency Treebank for training and testing. In an ideal situation every presented number would be measured on the same set of data. Yet the thesis presents outcomes of several years of research and unfortunately, the PDT version 1.0 was not available at the beginning. Some old figures that were measured on smaller data cannot be re-measured on PDT 1.0 because we cannot recover the exact state of the parser from those days. However, all the data sets are parts of evolving PDT, they are thus domain-compatible and the results should be comparable to some extent.

The table below gives an overview of the data sets and their sizes. Note that only PDT 0.5 and PDT 1.0 are official releases of the treebank. The others are technical labels of data that were available at various moments.

PDT 0.5 used two test sets, d-test and e-test. D-test stands for "development test data" and it is the default whenever we do not state otherwise. E-test was used only once in 1998 for cross-evaluation of the results (see Section 7.9). See also Hajič (1998); Hajič et al. (1998).

All results achieved after 1999 come from PDT 1.0 (Böhmová et al. (2000); visit http://ufal.ms.mff.cuni.cz/pdt/Corpora/PDT_1.0/Doc/PDT10_data.html for data layout). Its test set contains 8159 sentences and 126030 word/punctuation tokens. The training set contains 81614 sentences and 1255590 words.[6] Of course the respective training and test sets do not overlap. Note however that the PDT 0.5 test set cannot be used to evaluate a parser trained on the PDT 1.0 training set since the old development test data have been recycled as training data in PDT 1.0.

Finally, we defined a partitioning of the PDT 1.0 training data, roughly in the 9:1 ratio, so we got new training and test data, which both contained manual morphological

[5] This is the Czech sentence for testing font appearance (because it contains all letters of the alphabet). Its usual English counterpart is the sentence "The quick brown fox jumps over the lazy dog."

[6] Only 7319 of the test sentences and 73088 of the training ones are non-empty, so the average sentence length is 17 tokens. Maximum number of tokens in one sentence is 100 tokens for the test data, and 194 tokens for the training data.

annotation. Such data could be used to evaluate taggers as well as for running some interesting experiments that need the manual data.

Version	Training			Testing			Remarks
	Files	**Sentences**	**Words**	**Files**	**Sentences**	**Words**	
PDT 0.1	21	1050	18015	2	100	1590	Apr 1997
PDT 0.2	21	1050	18015	4	199	3036	Apr 1998
PDT 0.3	300	13481	230450	81 (*odd-numbered* bcc, bmc, bmd, bv1)	3697	63718	Jul 1998
PDT 0.5	437 (bc[bde]*, bl[12abcd]*, bm[12ab]*, bva*)	19126	327597				Aug 1998
PDT 0.5 e-test				81 (*even-numbered* bcc, bmc, bmd, bv1)	3787	65390	
PDT 1.0	1583 (c*, l101-lt52, m*, v*)	81614 (73088)	1255590	153 (l[uvw])	8159 (7319)	126030	2000
PDT 1.0c1	44 (c1*)	2207	30450				
PDT 1.0 mtrain / mtest	1425	65847	1133509	158 (every 10th from 1.0 train)	7241	122081	Test has manual morphology

6 The Baseline

There has been an attempt to model syntactic dependencies in Czech, described in Zeman (1997) and Zeman (1998). That model was quite unsuccessful in terms of accuracy but we can use it as something to compare new accuracies to. We also will briefly summarize the model architecture because its core remains essentially unchanged in the improved models.

The baseline dependency model resembles a bigram language model. In a bigram language model, statistics are collected about pairs of words that occurred adjacently in a text corpus. Relative frequencies of such pairs provide a maximum-likelihood estimation of the probability of the next word in a sentence, given the previous words. Using the probabilities, a language model predicts the next words and helps prune hypotheses about the sentence.

There is a strong (and false) assumption of statistical independence of the predicted word on any other word but the immediately preceding one. In fact there often are longer dependencies, obvious or hidden, but it is not feasible to reflect them all in the model, and it turns out that even *with* the assumption of independence a model can work quite well.

Another problem is that the model needs to know the current word to predict the following one — but the current word has been predicted as well and it may have been predicted wrongly. Usual workaround is to maximize the probability of the whole sequence of words (the sentence), defined as the product of the probabilities of each word given their predecessors. That way also right-side context can be considered. Some efficient method, like the Viterbi algorithm (Viterbi (1967); Manning and Schütze (1999: 332)), has to be employed to perform the maximization.

The dependency model is similar to the bigram language model in that it provides probability estimates for pairs of words. This time, however, the words in a pair are not necessarily adjacent in the sentence. Instead, dependencies are examined and pairs (w_i, w_j) collected where $d(w_i)=w_j$.

The sequential nature of a language model is suppressed although one could define an ordering for the dependencies. Nevertheless we still maximize the probability of the whole dependency structure rather than the probabilities of single dependencies alone. Analogously to the language model we assume that a probability of a word depending on another word depends solely on the two words.

Let $D(w_g, w_d)$ denote the event that somewhere in some sentence there is a dependency between two words, w_g (the governor), and w_d (the dependent). $P(D(w_g, w_d))$ is the probability of such event. We will call it the **dependency probability.** Note that it does not depend on the sentence in which the dependency occurred. The symbols w_g, w_d, respectively, shall be regarded as indices to the lexicon rather than indices to any particular sentence. That is why we have to distinguish the events $D(w_g, w_d)$ and $d(w_d) = w_g$ — the latter refers to a sentence.

Example: the dependency probability of *Česká republika*, "Czech Republic", estimated as the relative frequency in PDT 1.0, is $P(D(\text{"republika"}, \text{"česká"})) = 2.2\times10^{-5}$,

which corresponds to 56 occurrences in the corpus. Another example: *reprezentanti republik,* "representatives of the republics", occurred exactly once and *P(D("reprezentanti", "republik")) = 3.9×10^{-7}*.

Let the **tree probability** of the dependency structure (tree) *T,* built upon the underlying sentence *S*, be defined as follows:

$$P(T|S) = \prod_{i=1}^{|S|} P(D(d_T(w_i), w_i))$$

Recall the example sentence (8) (*S = "Studenti mají o jazyky zájem, fakultě však chybí angličtináři."* lit. "Students have in languages interest, faculty but miss teachers-of-English."). Its dependency structure was *T = "2,8,5,3,2,8,9,0,8,9,0"*. Note that it does not make sense to examine *P(T)* (without respect to *S*) since it would be probability of a bare structure without any lexical cast. We are rather interested in *P(T|S)*, the probability of the structure given the sentence: *P(T|S) = P(D("mají", "studenti")) × … × P(D("#", ".")) = 3.9×10^{-7} × 7.4×10^{-6} × 8.1×10^{-5} × 0 × 1.2×10^{-5} × 1.9×10^{-4} × 0 × 7.3×10^{-4} × 1.6×10^{-6} × 0 × 2.3×10^{-2} = 0*. Unfortunately, there is a serious problem of data sparseness: any tree is quite likely to contain a dependency not known from the training data, which will thus cause *P(T|S)* to be zero. Zeman (1997) addresses this issue by *smoothing*. We will come to it later in this chapter; for now, just to give an idea how small the tree probabilities are, assume that each dependency in *T* has been seen at least once (i.e. replace zero counts by ones, leave everything else). Then the tree probability *P(T|S)* would be *8.2×10^{-55}*.

Of course, the tree probability is closely related to the length of the sentence. If a sentence were 30 words long — not as unusual indeed — and its tree contained dependencies with the same mean dependency probability, the order of the tree probability would be *10^{-103}*. That does not matter though, because a parser always constructs a tree for a known sentence and it only needs to compare probabilities of the trees for that sentence — meaning all trees compared contain the same number of words.

Another observation is that a tree probability distribution is always deficient: there are sets of dependencies that are not valid dependency structures (like the cyclic set "4,1,2,3") but receive a non-zero probability. We can restrict ourselves to valid trees, yet most of them will be length-incompatible with our sentence. And the probabilities of the rest will sum up to something far below one because much of the probability mass will go to sentences with different lexical cast. All the probability leaks mentioned could theoretically be patched by normalizing the probabilities so they sum up to one. However, finding the normalizing factor would be expensive if not impossible, and we claim that it is not necessary if we compare probabilities of trees for *one* sentence. We only need to design the tree constructing procedure so that it outputs valid trees compatible with the given input sentence.

Given the above definition of tree probability, the baseline parser tries to find a tree with maximal probability given the input sentence:

$$DS(S) = \arg\max_T (P(T|S)) = \arg\max_T \left(\prod_{i=1}^{|S|} P(D(d_T(w_i), w_i)) \right)$$

The dependency probabilities $P(D(w_g, w_d))$ can be estimated by relative frequencies of the observations of such dependencies in the training data. The remaining questions are:

1 What word information should be used to match words and dependencies in the training and test data?

2 Which smoothing method should be applied to the probability distribution to avoid zero probabilities of unknown dependencies?

3 Given the dependency probabilities for any dependency compatible with the input sentence, how do we find the most probable dependency structure?

Zeman (1997) gives the following answers to the questions:

1 **Word matching.** The parser is *not* lexicalized; otherwise the data would prove much sparser. Two words match if they are assigned the same morphological tag. Two dependencies match if their governors match and their dependents match. To give an example, consider the phrases *evropské země* ("European countries") and *zelené myšlenky* ("green ideas"), both tagged as AAFP1----1A---- NNFP1-----A----. An observation of *any* of these two phrases will be counted as an occurrence of the same dependency, and the two dependencies will get the same probability, despite the fact that the former is quite likely to appear in a newswire text while the latter is semantically problematic and will hardly appear anywhere.

The usage of morphological tags implies yet another question: which source of morphological information do we use? Human annotation, tagger output, or ambiguous morphological analysis? Zeman (1997) uses the ambiguous MA — anyway, there were not sufficient resources for disambiguated data in 1997. He uses fraction frequencies: if he learns a dependency whose governor allows 3 tags and the dependent allows 2 tags, he views the combinations of the tags as six dependencies, each seen 1/6-times. Later as the parser needs to know the probability of such dependency, it simply sums up the probabilities of the possible combinations. In the following equations, $C(D)$ denotes the number of occurrences of the event D in training data, and C is the number of words of training data:

$$C\left(D\left(\left[t_1 / t_2 / t_3\right],\left[t_4 / t_5\right]\right)\right) = \sum_{\substack{i=1\ldots3 \\ j=4\ldots5}} \frac{C\left(D\left(t_i, t_j\right)\right)}{C}$$

2 **Smoothing.** A small constant is added to each dependency probability. Formally the probability $P(D(w_g, w_d))$ is linearly interpolated between the relative frequency of the dependency, and the uniform probability of a dependency (1/number of possible different dependencies in Universe). 1786 is the number of morphological tags defined for Czech in 1997:

$$P\left(D\left(w_g, w_d\right)\right) = 0.99 \times \frac{C\left(D\left(w_g, w_d\right)\right)}{C} + 0.01 \times \frac{1}{1786^2} = 0.99 \times \frac{C\left(D\left(w_g, w_d\right)\right)}{C} + 3.1 \times 10^{-9}$$

3 **Searching.** A greedy algorithm was used. It was fast but it did not guarantee that the optimal tree was found. In each round, the parser considered the set of allowed dependencies. A dependency was allowed, if its governor had already been added to the tree and the dependent had not. The tree was thus constructed

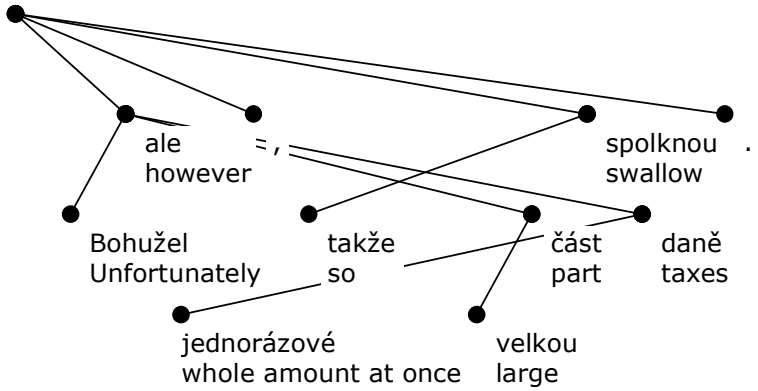


Figure 6: Bohužel ale jednorázové, takže velkou část spolknou daně.
Unfortunately (we got the whole amount) at once so taxes will consume a big part of it.
[bad parse]

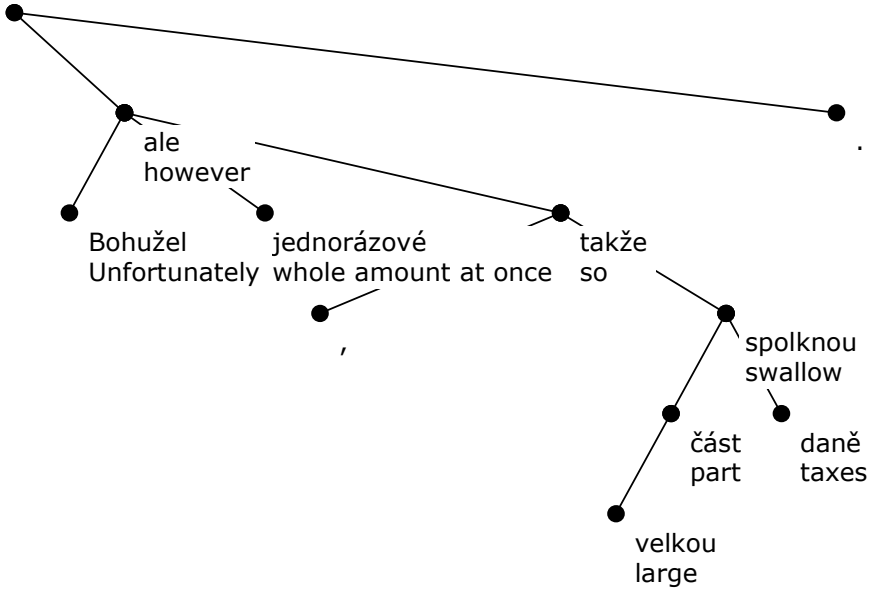


Figure 7: Bohužel ale jednorázové, takže velkou část spolknou daně.
Unfortunately (we got the whole amount) at once so taxes will consume a big part of it.
[correct parse]

from the root to the leaves. Among the allowed dependencies, the parser always selected the most probable one and added it to the tree. After the n^{th} round the tree was complete.

According to Zeman (1997), the standard dependency accuracy of the model described above is 31 % (measured on PDT 0.1). Figure 6 shows an example output of the model of Zeman (1997). It is a rather unsuccessful (A = 40 %) attempt to analyze the sentence

(11) *Bohužel ale jednorázové, takže velkou část spolknou daně.* "Unfortunately [we got the whole amount] at once so that a big part will be swallowed by taxes."

The correct parse of the sentence (11) is shown in Figure 7.

To give an idea just how bad is the accuracy we finally show the accuracy of some naïve parsers not employing any model. This time the accuracy is counted on PDT 1.0 test set.

The most naïve parser assigns a random dependency to each word, except of the last word in the sentence, which always depends on the root. The dependencies of the other words only observe the condition that a dependency structure cannot contain cycles. We conducted five consecutive experiments with the random parser. The accuracy was 10.55, 10.66, 10.64, 10.60, and 10.67, respectively. The mean accuracy was 10.62 % with a standard deviation of 0.04. We show some examples of random parses in Figure 8.

Slightly better is a parser that assigns the zero (root) dependency to each word, building an **umbrella-style tree:** 11.6 %.

In contrast, the improvement brought by **chain-style trees** is much more significant. In a **right-branching** chain-style tree each word depends on the preceding

Figure 8: Arafat called for reforms. [random parses]

one. 24 % of such dependencies are correct on average. In a **left-branching** chain-style tree each word depends on the next one, only the last word (usually punctuation)

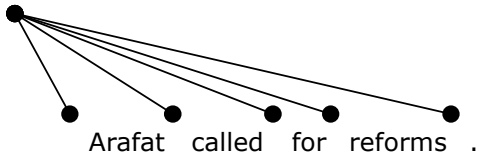
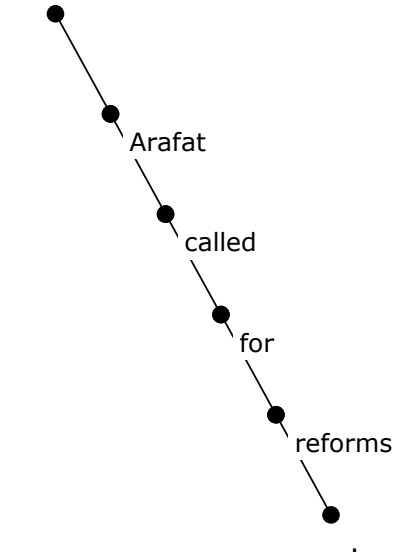
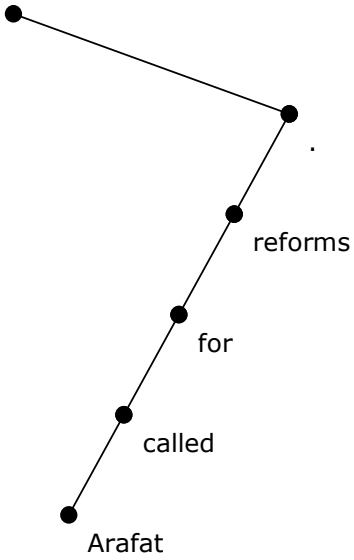


Figure 9: Arafat called for reforms. [umbrella parse]

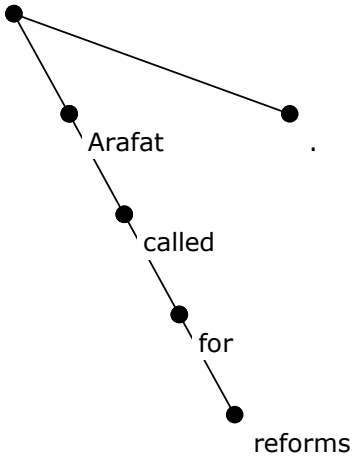
depends on the root. The accuracy of left-branching chains is 28.6 %. Figure 10 shows examples of chain trees.



a) Right-branching chain-style tree



b) Left-branching chain-style tree



c) Reversed-one-style (R1) tree

Figure 10: Arafat called for reforms.
[chain parses]

Finally, the best performing naïve method is the one producing **reversed-one-style (R1) trees** (Figure 10c), i.e. right branching chains with the last word always depending on the root. The R1 trees are 30 % accurate, which is only one per cent point below the published accuracy of the parser of Zeman (1997).

The optimistic side of this degrading comparison is that such results can only be improved. We show in the following chapters how including more linguistics into the model increases the accuracy while still using the same statistical core.

7 Model One

The first model proposed by this thesis is an improved version of Zeman (1997). The statistical core remains the same but some constraints (most notably the projectivity constraint) have been implemented. Also, the model has been augmented by some new statistics, like fertility, word adjacency, and dependency direction. These add-ins were first described in Hajič et al. (1998)[7]; we describe them in the following subsections.

The Model One was trained and tested on the PDT 0.5 data (see Chapter 5) and its maximal accuracy (i.e. with the optimal setup) is 55 % (Hajič et al. (1998))[8]. Most of the changes made on the way from the Baseline Model to the Model One are language independent (we will explicitly point out the exceptions). In these terms the model remains as much data-driven as possible.

7.1 Enforcing projectivity

Having seen the parse in Figure 6 we realize that there is absolutely no correlation between the dependencies and the word order. Although Czech is usually classified as a free word order language, its word order indeed is not *that free*. It may seem difficult to pose a constraint on the word order as phrases of arbitrary length can be inserted almost anywhere in the sentence. Nevertheless, there is a characteristic of a sentence that combines the structure with the word order: it is the notion of **projectivity.**

Several mutually equivalent definitions of projectivity have been proposed. For one of the early ones see Marcus (1965); for more details on projectivity and non-projective constructions, refer to Chapter 12. One of the definitions follows.

A dependency A-B (where A is the governing node) is **projective** if and only if all the words that are placed between A and B are included in the subtree of A. If the tree is displayed so that the x-coordinates of nodes correspond to the word order (i.e. it is displayed the way described in Section 4.2), each non-projective dependency will cross at least one perpendicular from another node (it will not necessarily cross another dependency.)

Projectivity is an important attribute of the relation between the word order and the dependency structure. It does not depend on the length of the phrases but it still allows us to distinguish "normal" sentence structures from the "wild" ones. We deliberately do not call them "incorrect" because not all non-projective dependencies are errors in Czech.[9]

A classical example of a non-projective sentence is the Czech

---

[7] Most of the Model One is outcome of a research conducted before and during the Workshop 98 organized by the Johns Hopkins University, Baltimore, MD, and funded by the National Science Foundation.

[8] Besides the PDT 0.5 *development test* data described in Chapter 5, there also were *evaluation test* data of a similar size, that were first available to us after the work on Model One had been finished. The accuracy measured on the evaluation test data was 56 %.

[9] Note however that non-projective constructions cannot be described by an ordinary context-free grammar.

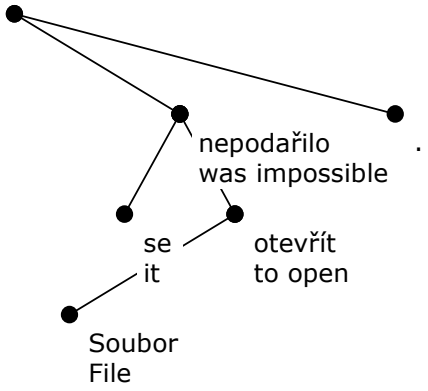


Figure 11: Soubor se nepodařilo otevřít.
The file could not be opened.

(12) Soubor se nepodařilo otevřít. (Lit. "File it did-not-succeed to-open.")

The Figure 11 shows that the dependency *otevřít soubor* "open a file" is non-projective.

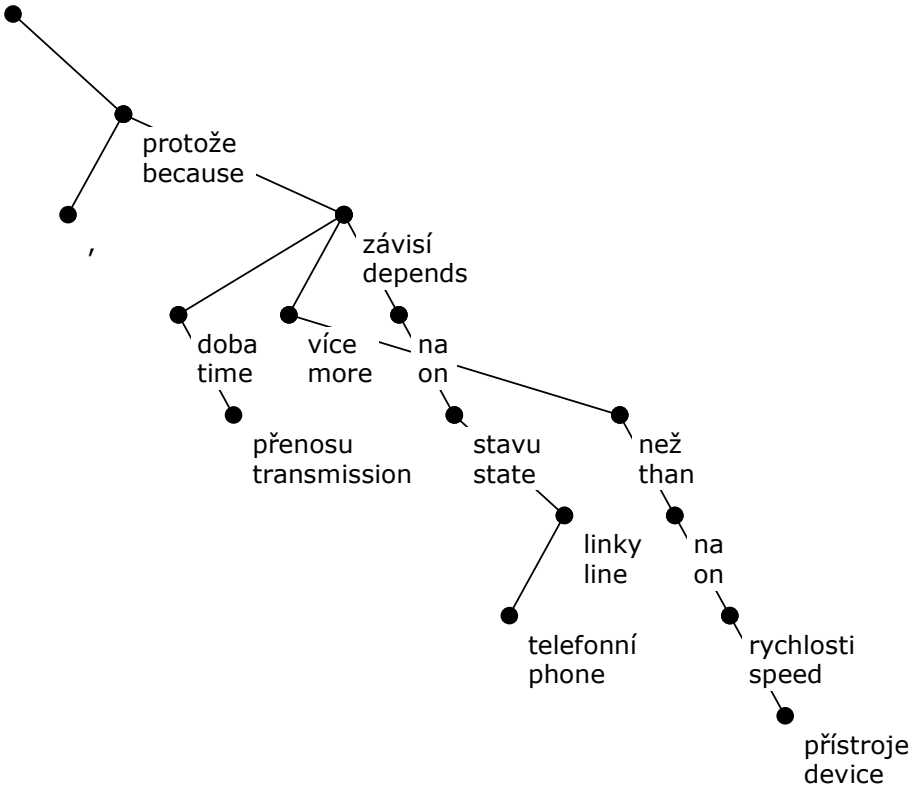


Figure 12: , protože doba přenosu více závisí na stavu telefonní linky než na rychlosti přístroje
because the transmission time depends on the state of the carrier rather than on the speed of the device

Another example is given in Hajič et al. (1998) (a sentence fragment):

(13) , protože doba přenosu více závisí na stavu telefonní linky než na rychlosti přístroje (Lit. "because time of-transmission more depends on state of-

phone line than on speed of-device") "because the transmission time depends on the state of the carrier rather than on the speed of the device." (see Figure 12)

Nevertheless such dependencies are quite rare: Hajič et al. (1998) counted only 1.8 % of dependencies in PDT 0.5 being non-projective,[10] so it is useful to require that all the dependencies generated by the parser be projective. This is particularly true until the overall accuracy of the parser is less than 90 %. In Section 12 we will address the issue of identifying and solving the cases of correct non-projective constructions.

Projectivity of a dependency structure can be enforced by either connecting at each step only neighboring components (in certain configurations) in a forest, or simply by proposing a new dependency and then checking whether it would be non-projective (the latter is obviously less efficient but it is also less restrictive as to the order in which the dependencies are constructed).

Note: One may argue that giving precedence to the projective trees is a language dependent feature since there might be languages with a consistently non-projective word order. We however claim that there is a significant number of languages with rare or no non-projectivity at all, which makes this part of our parser reusable.

7.2 Reduction of the tag set

Some of the morphological information encoded in the tags has no influence on syntax. For instance, all adjectives, verbs and adverbs have affirmative and negative forms whereas the negative form is constructed completely regularly by simply inserting the prefix *ne-* to the word beginning. The negative forms behave syntactically almost the same way the affirmatives do.[11] So negativeness has little or no influence on Czech syntax.

Now we will show how using the negativeness information can damage the statistical model. As an example, consider the phrase *měl by* "would have" or "should". It occurs 412 times in PDT, while its negative counterpart, *neměl by* "should not" occurs only 45 times, i.e. nine times less! Since the Model One is not lexicalized, we should convert the phrases to morphological tags: VpYS---XR-**A**A--- Vc-X---3------- corresponds to the affirmative case and VpYS---XR-**N**A--- Vc-X---3------- corresponds to the negative case. The dependency between the former two tags occurred 1589 times in PDT, the latter occurred 192 times (eight times less). If we train the Model One on the original tag set, the negative dependencies will be at a disadvantage when

[10] About 79.4 % of all trees have all dependencies projective. The number of trees that contain one or no crossing (non-projective) dependency is 93.8 % and the number of trees with at most two such dependencies is 98.3 %. Recently we recounted the non-projective constructions on the training portion of PDT 1.0. There were 23691 crossing dependencies, i.e. 1.9 %. 56168 sentences (76.8 %) were all-projective.

[11] The exception is phrases like *nikdo nepřišel* "nobody came" (lit. "nobody not-came"). Unlike English, Czech requires a negative subject be followed by a negative predicate, so **nikdo přišel* would be incorrect. However, most subjects are expressed by nouns that are negativeness-neutral (*Martin nepřišel* "Martin did-not-come" is OK) and thus sentences with double negation are relatively rare. If they appear they can still be covered by the more general statistics that do not distinguish negativeness.

compared to the rest. During parsing, negative dependencies will not be built if there is another option. An intuitive solution is to merge the tags VpYS---XR-**A**A--- and VpYS---XR-**N**A--- into one, say VpYS---XR-**X**A---.

There are other options of reducing the information contained in the tags. The following table summarizes all reduction types done in Model One.

Reduction type	**Example tags before**	**Example tags after**	**Example words before**	**Example words after**
discard negativeness of nouns, adjectives, verbs, and adverbs	NNMS1-----N AAMS1----1N VB-P---1P-NA Dg-------1N	NNMS1-----A AAMS1----1A VB-P---1P-AA Db	*nehezký* "not-lovely" *nedělám* "I-don't"	*hezký* "lovely" *dělám* "I-do"
discard comparison degree of adjectives and adverbs	AAMS1----2A Dg-------3A	AAMS1----1A Db	*lepší* "better" *dříve* "earlier"	*dobrý* "good" *právě* "just"
discard stylistic nuances	AAMS1----1A---6	AAMS1----1A	*zelenej* "green-(colloquial)"	*zelený* "green"
convert possessive adjectives to normal adjectives	AUMS1M	AAMS1----1A	*otcův* "father's"	*zelený* "green"
discard tense of deverbative adjectives	AMMS1-----A	AGMS1-----A	*dodělavší* "having-finished-to-do"	*dělající* "doing"
convert special adjectives to normal adjectives	AOYS	AAYS1-----A	*svůj* "oneself's"	*zelený* "green"
convert short forms of adjectives to normal adjectives	ACYS------A	AAYS1-----A	*stár* "old"	*starý* "old"
convert abbreviated adjectives to normal adjectives	A.	AAMSX----1A	*zel.* "gr."	*zelený* "green"

Reduction type	**Example tags before**	**Example tags after**	**Example words before**	**Example words after**
normalize forms of pronouns	PHZS3--3	PPZS3--3	*mu* "him"	*jemu* "him"
	P5ZS2--3------1	P5ZS2--3	*něho* "him" gen.	*něj* "him" gen.
	P7-X3	P6-X3	*si* "himself" dat.	*sobě* "himself" dat.
convert possessive pronouns to adjectives	PSIS4-S1	AAIS4----1A	*můj* "my"	*celý* "whole"
	P8MS4	AAMS4----1A	*svého* "oneself's" acc.	*jiného* "other" acc.
convert demonstrative pronouns to adjectives	PDMS4	AAMS4----1A	*tohoto* "this" acc.	*jiného* "other" acc.
convert "type L" pronouns to adjectives	PLMP1	AAMP1----1A	*všichni* "all"	*dobří* "good"
convert relative pronouns to nouns	PJMS4	NNMS4-----A	*jehož* "whose" acc.	*zákazníka* "customer" acc.
	PE--1	NNMS1-----A	*což* "what"	*pán* "man"
convert relative pronouns after prepositions to personal pronouns	P9FS2	P5FS2--3	*níž* "whom" gen.	*ní* "her" gen.
convert relative possessive pronouns to adjectives	P1IS4FS3	AAIS4----1A	*jejíž* "whose" acc.	*celý* "whole" acc.
convert substantive pronouns to nouns	PKM-1	NNMS1-----A	*kdo* "who"	*pán* "man"
	PQ--1		*co* "what"	
	PZM-1		*někdo* "somebody"	
	PZ--1		*něco* "something"	
	PWM-1		*nikdo* "nobody"	
	PW--1		*nic* "nothing"	
convert attributive pronouns to adjectives	PZFS1	AAFS1----1A	*leckterá* "all sorts of"	*skutečná* "real"
	PWMP1	AAMP1----1A	*žádní* "no"	*dobří* "good"

Reduction type	Example tags before	Example tags after	Example words before	Example words after
convert numerals	ClXP3---------1	ClXP3	*set* "hundred" dat.	*dvěma* "two" dat.
convert interrogative numerals to cardinals	C?--1	Cn-S1	*kolik* "how many"	*padesát* "fifty"
convert indefinite numerals to cardinals	Ca--1		*několik* "a few"	
convert ordinals to adjectives	CrMS1	AAMS1----1A	*třetí* "third"	*soukromý* "private"
convert indefinite numerals to adjectives	CwIS4	AAIS4----1A	*nejeden* "several"	*celý* "whole"
convert generic numerals to adjectives	CdIS4	AAIS4----1A	*dvojí* "twofold"	*celý* "whole"
convert numerals to adjectives	ChMP1	AAMP1----1A	*jedni*	*dobří* "good"
convert multiplicative numerals to adverbs	Cv	Db	*čtyřikrát* "four times"	*snad* "perhaps"
convert interrogative multiplicative numerals to adverbs	Cu	Db	*kolikrát* "how many times"	*již* "already"
convert fractions to nouns	CyFS1	NNFS1-----A	*třetina* "one third"	*skutečnost* "reality"
convert Roman numbers to cardinals	C}	C=	*II* "II"	*15* "15"
convert imperatives to present tense indicative	Vi-S---2--A	VB-S---2P-AA	*rozuměj* "understand"	*dostaneš* "you get"

Reduction type	Example tags before	Example tags after	Example words before	Example words after
discard tense of transgressives	VmXP------A	VeXP------A	*semnuvše* "having rubbed"	*věříce* "believing"
discard vocalization of prepositions	RV--2	RR--2	*ze* "from"	*z* "from"

On the other hand we considered the information in punctuation tags insufficient, as it is often necessary to distinguish between the final stop, the comma, and other punctuation marks (for instance the parentheses). So we combined the punctuation tag with the "lemma". The tag "Z:", originally denoting any punctuation, remained untouched only when denoting a colon. If it belonged to a comma, for instance, it changed to "Z,"; if it modified a period, it changed to "Z.". A final punctuation of the sentence was further augmented by the letter K: Z.K, Z?K, Z!K…

Before the reduction (or more properly: the modification) of the tag set there were 1279 tags present in the training part of PDT, out of 4288 possible. After the reduction this number decreased to about 452 so the reduction rate was approximately 65 %.

The way of modifying tag set described above is the only language and theory dependent component of the Model One. Hajič, Collins, and Ramshaw in Hajič et al. (1998) have tested a language independent alternative. They applied a clustering algorithm (see e.g. Manning and Schütze (1999), Chapter 14) to the set of morphological tags. We will further discuss modified tag sets in Section 8.4.

7.3 Lexicalization

Chapter 6 defines word matching on the basis of the m-tags assigned to the words. Such model is called **non-lexicalized**. If it used lemmas or word forms to match words, it would be **lexicalized**. Obviously there are lexically determined dependencies that could be caught much easier in a lexicalized model. Some notorious examples would be verb arguments, some preposition-noun pairs, or idiomatic phrases in general. The expectations laid on parser lexicalization are further supported by the results reported for English parsers: once they got lexicalized, their performance improved significantly (about 10 % points — see also Charniak (1997)).

The lexicalized version of the Model One uses lemmas (automatically assigned by a lemmatizer) for word matching and backs off to m-tags whenever there is no evidence of a given word-on-word dependency. The backing off is done using linear interpolation of the two probabilities, one estimated by the lexicalized model and the other estimated by the m-tag model.

$$p = \lambda p_{lex} + (1-\lambda) p_{mtag}$$

We tested both lemmas and word forms[12] as the word matching attributes for the lexicalized model. Lemmas are more ambiguous than word forms. On the other hand, forms are much sparser than lemmas. In the electronic dictionary used for PDT annotation (see Hajič (2004)) there are about 700K lemmas and about 20M forms, which yields an average of enormous 28.6 forms per lemma. Such ratio naturally cannot be rendered in real data but there are still more than twice as many forms as lemmas: the PDT 0.3 training data contained 21647 lemmas and 58347 word forms, which yields a ratio of 2.7 forms per lemma. Data sparseness is affected by the fact that most of the extra words have been observed only once, as seen in the following table.

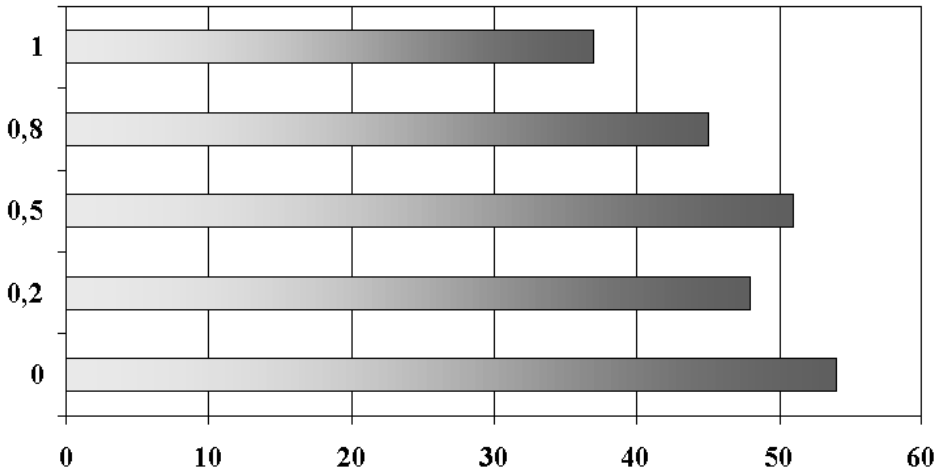
Occurrences in PDT 0.3 train	**Lemmas from dictionary**	**Lemmas from lemmatizer**	**Lemmas from human**	**Word forms**
≥1000	24	25	24	19
≥100	217	221	225	145
≥50	528	530	528	364
≥25	1207	1208	1207	879
≥5	5165	5051	5071	5964
>0	23474	21711	21647	58347

The probability distribution of the lexical model (lemmas by a lemmatizer) on PDT 0.3 training data:

Number of different dependencies	**Maximum possible (uniform) entropy**	**Entropy**	**Perplexity**
118827	16.86	15.22	38083

We tested a bunch of values of λ. Unfortunately, we obtained the best accuracy using $\lambda=0$, which means the lexicalization did not help. The chart below displays accuracies for various values of λ.

[12] Word forms have to be converted to lowercase, otherwise the statistics get split unnecessarily.



To figure out how much the problem was caused by data sparseness, we considered all dependencies seen five times or less **unknown** (the number five has been found empirically). To distinguish unknown dependencies from impossible ones, if a lexical dependency was unknown, the λ was set to 0. If the dependency was unseen at all, the same setting as for dependencies seen six or more times was used: λ=0.5. This modification improved the accuracy to 54 %.

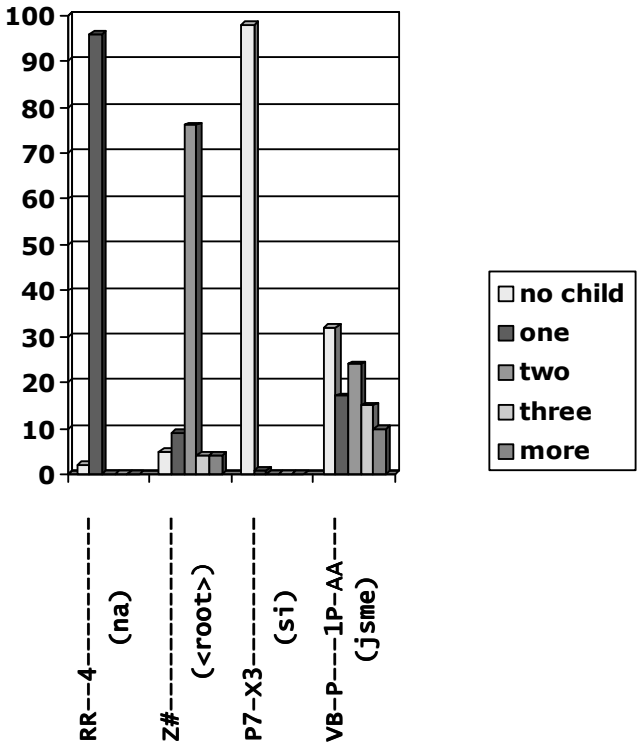


Figure 13: Varying fertility of some m-tags: a preposition (one child in 96 %), the sentence root (two children in 76 %), a reflexive pronoun (childless in 98 %) and a verb (no clear preference).

7.4 Modeling word fertility

Some words tend to have many dependents while some others are almost all the times leaves. And some words usually have an exact number of children, for instance prepositions have exactly one: a head of a noun phrase. The parser of Zeman (1997) did not reflect this observation and if there was one preposition and several nouns in a sentence, it usually hung all the nouns under the single preposition. To avoid this, the Model One contains a part which knows how often a given tag is a leaf, how often it has one child, how often two, and how often three or more. In Hajič et al. (1998) this feature is called *valency* but *fertility* is probably a better term. The parser multiplies the estimation of dependency probability by the probability that the governing node has *n+1* children where *n* is the current number of children. Whenever run on tags coming from the ambiguous morphological analysis, it computes the average fertility over all ambiguous tags of the governing node.

7.5 Governor-dependent adjacency

Another way of taking the word order into account is to record separately dependencies of different length. The length is the distance between the governor and the dependent in the sentence, in words. If the nodes are adjacent, the length of the dependency is 1. The Model One sorts dependencies into two classes: adjacent and non-adjacent.

For example the dependency D(NNFS2-----A----, AAFS2----1A----) (a feminine singular genitive adjective depending on a feminine singular genitive noun, e.g. *maďarské měny*, "Hungarian currency") prefers adjacent nodes. In PDT 1.0c1 it occurred 153× with adjacent nodes and only 8× with non-adjacent ones. Collecting distance statistics should prevent the parser from attaching arbitrarily distant adjectives to a noun.

There are dependencies having the opposite preference as well, although that could be caused by the fact that their tags appear rarely next to each other in general (whether or not they depend one on the other). An example is D(NNNP2-----A----, C=----------) (a cardinal number depending on a neuter plural genitive noun, e.g. *17 let* "17 years"). In the same portion of PDT this dependency occurred 27× with distant nodes and 6× with adjacent ones.

7.6 Dependency direction

Even more important than adjacency is the direction of the dependency. The Model One records separately dependencies going left-to-right (the governor precedes the dependent in the sentence), and those going right-to-left (the dependent precedes the governor).

The typical right-branching dependencies include nouns on prepositions (for instance *v části* "in part"; the corresponding dependency D(RR--6----------, NNFS6----------) led 245 times left to right, zero times right to left). Similar is the case of subordinating conjunctions (for instance *že jde* "that he goes"; the corresponding dependency D(J,---------- VB-S---3P-AA---) led 101 times left to right, zero times right to left). Our last example is the infinitive depending on a modal verb: *nemůže zveřejnit* "she cannot publish"; the corresponding dependency D(VB-S---3P-AA--- Vf--------A----) led 168 times left to right and 11 times right to left.

A typical left-branching dependency is a noun governing an adjective, both agreeing in gender, number and case: *ruské vlády* "Russian government". D(NNFS2-----A---- AAFS2----1A----) went 161× right to left, never left to right.

7.7 Proportional training

This section discusses an issue that we hoped might help the parser in case there is no tagger available and the morphological tags are not disambiguated.

As we mentioned before, the ambiguous tag combinations over an edge get a uniform distribution of probabilities. However, some of them are more probable than the other. The model may reflect this because particular tag combinations may have been present in various edges. But this is only an indirect binding and we wanted to emphasize the differences between particular tags, some of them being very frequent while the others were hardly to see at all.

So we built a small unigram probability distribution for the tags and then, training the tree structures, instead of increasing the frequency of each tag combination by *1/n* we weighted the frequency by the probability of the tags in the given combination. Of course, it was normalized so that the values for all combinations in one edge summed

to 1. However, this feature did not improve the results at all, they were even worse than before. So we omitted the proportional training in ongoing research.

7.8 Searching for the best tree

Unlike the baseline model, which used a greedy algorithm to approximate the optimal tree, the Model One employed a Viterbi-inspired beam search. It was a compromise between the greedy approach that did not guarantee it would find the optimum, and the "British Museum approach" that would go through the exponential space of all possible trees and… never end.

The beam searching method records *N* best partial trees after each round (*N* is the width of the beam). In the next round the greedy algorithm would find the most probable dependency and add it to the tree. The beam search, on the other hand, finds *N* or more best dependencies and creates *N* best new partial trees. It may gradually add all the new dependencies to the same tree from the previous round and discard the other trees. Or it may add each dependency to a different tree, or it can do any mixture of the two extremes. In fact, the algorithm finds *N* best new dependencies and gradually adds each of them to each of the trees from the previous round, creating *N*×*N* new trees. It discards duplicates[13], keeps the *N* best new trees and removes the other. Should there

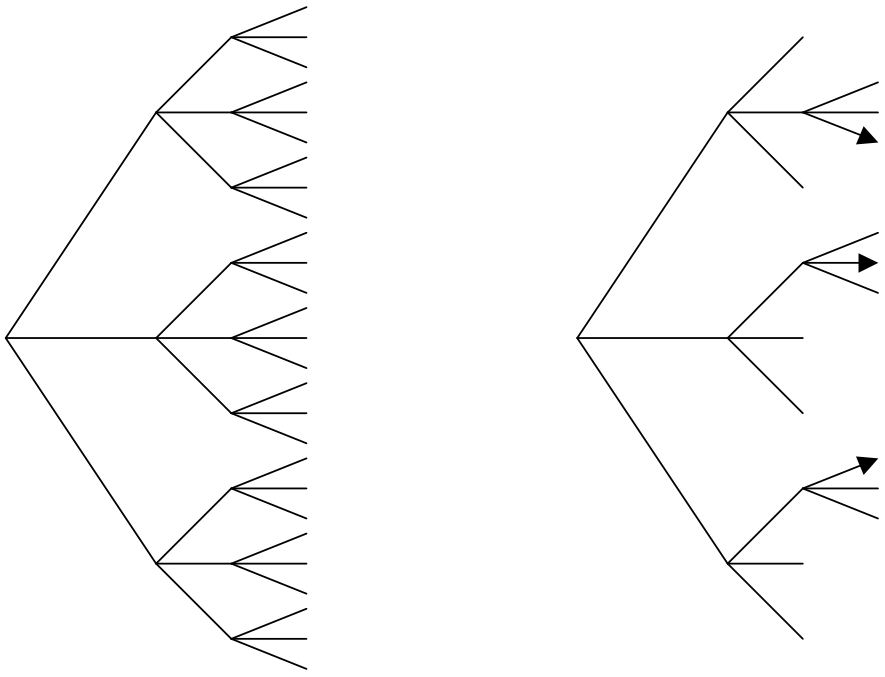


Figure 14: Search with a beam width of 3. The vertices represent states of the analysis and the corresponding partial trees.

[13] Duplicates are frequent. Consider this example: Round *i*, tree X is simultaneously extended with dependencies *a* (the best one in terms of probability) and *b* (the second best). Two new trees X_a and X_b arise and advance to the next round. Round *i*+1, tree X_a is extended with *b* — of course, if it was the second best in last round, it becomes the best after *a* has been added. A new tree X_{ab} arises. Similarly, X_b is extended with *a* to create X_{ba}. However, X_{ab} and X_{ba} denote the same tree, so one of them must be removed.

be less than *N* trees after discarding duplicates, more than *N* new dependencies can be considered.

The effect of the beam search on the parsing accuracy is discussed in Section 7.9.

7.9 Evaluation of Model One

7.9.1 Contribution of particular features

Some of the features of Model One immediately and significantly improve the results while some others work only when combined together. For this reason it is not always possible to specify the effect of a feature in terms of one figure. We rather present a table of feature combinations and the achieved accuracy levels. The baseline result (repeated from Chapter 6) comes from PDT 0.1 test set (100 sentences), the rest from PDT 0.2 test set (199 sentences).

Legend:

The column labeled N shows the number of correctly assigned dependencies, out of the 3036 total. The column labeled A shows the accuracy rate in %.

prj Enforced projectivity of all dependencies.

vit The Viterbi 5-best search was used instead of a greedy algorithm (searching for a tree which has the highest possible product of probabilities of its edges).

red The reduced set of morphological tags was used both for training and parsing.

fer The fertility model was used.

prt Proportional training was used.

adj Adjacency (distance). A dependency is identified by the m-tags of the nodes and by the information whether the two words are or are not next to each other in the sentence.

dir Direction. A dependency is identified by the m-tags of the nodes and by the information whether governor precedes dependent in the sentence, or vice versa.

Note that not all possible feature combinations are included in the table.

Model	**N**	**A**
baseline	N/A	31.00
prj+prt	1226	40.38
prj+fer+prt	1235	40.68
prj	1237	40.74
prj+vit+prt	1241	40.88
prj+red+fer	1241	40.88
prj+red+prt	1242	40.91
prj+vit+red+prt	1253	41.27
prj+red	1256	41.37
prj+vit+red	1256	41.37
prj+red+fer+prt	1258	41.44
prj+vit	1259	41.47
prj+vit+fer	1271	41.86

prj+vit+fer+prt	1313	43.25
prj+vit+red+fer	1360	44.80
prj+vit+red+fer+prt	1361	44.83
prj+vit+red+prt+fer+adj	1458	48.02
prj+vit+red+fer+adj	1472	48.48
prj+vit+red+prt+fer+dir	1492	49.14
prj+vit+red+fer+dir	1505	49.57
prj+vit+red+prt+fer+adj+dir	1599	52.67
prj+vit+red+fer+adj+dir	1620	53.36

The last row of the table shows the best accuracy of the Model One on PDT 0.2. The same figure on PDT 0.5 is 51.5 % (d-test), and **54.1 %** (e-test). Unfortunately the above table was filled at the time when only version 0.2 of data was available.

An interesting perspective of the results was achieved by evaluating separately two parts of the PDT 0.5 test data. The training set and both the test sets contained texts from three different sources: from daily newspapers (*Lidové noviny* and *Mladá fronta Dnes*), from a business weekly (*Českomoravský Profit*), and from a scientific magazine (*Vesmír*). It turned out to be much more difficult to parse the last one (mainly because of the sentence length) than the former two. In the following table, the evaluation of the scientific magazine is labeled "scientific", whereas the data from the two other sources bear the label "normal".

D-test			**E-test**		
normal	scientific	**all**	normal	scientific	**all**
54 %	48 %	**51 %**	57 %	51 %	**54 %**

The Model One parser was implemented in C++ and it employed disk-mapped B-trees to record the huge dependency statistics. It was being run on a 686 Intel equipped with Linux. The processing of the 3697 sentences took about $3\,{}^{1}/_{2}$ hours, i.e. an average sentence took 3.4 seconds. Training on PDT 0.5 data (19126 sentences) took about 12 minutes (38 ms per sentence).

The accuracy further improves by 1 % if the beam width changes from 5 to 50. This of course lengthens the run time by a factor of 10 (34 seconds a sentence, 35 hours whole test data). The 55 % accuracy was the best we were able to achieve with the Model One.

7.9.2 Different sources of morphology

The baseline parser and all the parsers evaluated in Section 7.9.1 worked with sets of ambiguous m-tags that have been assigned to each word by the morphological analyzer (MA). This is not the only option of where to get the necessary morphological information. The other possibility is to employ a tagger and to use its output. Both

sources are noisy and can cause parsing errors. MA almost[14] always provides the correct annotation but the correct one is usually accompanied by a number of bad ones without any clue on which to prefer. A tagger on the other hand provides only one annotation but it is not always correct. There is a legitimate question, which of the two noises hurts the syntactic analysis more.

First experiments to answer this question for the Model One were run on PDT 0.3 in summer 1998 (Hajič et al. (1998)). Different sources of morphology were used for training and testing: dictionary, tagger, and human. Human manual annotation could not be used for testing, as it would not be available in a real use of the parser (and the test data did not contain human annotation anyway). However, it may have proven useful to train on human morphology even when parsing with a tagger or dictionary.

Training	**Testing**	**Accuracy**
13481 sentences	3697 sentences	
dictionary	dictionary	51.42
human	human	(51)
human	dictionary	52.59
human	tagger	53.44
dictionary	tagger	53.72
tagger	tagger	54.08

Since the test data are not annotated manually, we evaluated the human-human combination on a small set of 124 sentences cut off the training data (of course, this portion was not used to train the parsers involved in the experiment). Surprisingly, the human-human combination was by more than 3 percent points worse than the tagger-tagger combination. This might have been an effect of overtraining: the parser training dealt with some rare constructions that caused some errors in the average test data. Such strange constructions were however hidden by the tagger since it was not able to recognize them.

The better performance of the tagger-tagger combination over the human-tagger one can be explained more easily. The parser probably learned how to deal with the errors the tagger does. The similarity of the training and the test data turned out to be much more important than the accuracy of its morphological annotation (when the tagger accuracy is at least 92 %, which is the case here).

The following table shows the differences in probability distributions for the three morphology sources, measured on PDT 0.3 training data.

	Number of different dependencies	Maximum possible (uniform) entropy	Entropy	Perplexity
human	16163	13.98	11.30	2523
tagger	17391	14.09	11.42	2734
dictionary	82525	16.33	12.65	6442

[14] Of course the coverage of the dictionary is not 100 %.

8 Model Two

In the beginning of 2002 the parser was reimplemented in Perl. We call this new implementation "Model Two". Some features of Model One have been revised in Model Two and many new features have been added.

Model Two was being developed in the PDT 1.0 era. It was thus trained and tested mostly on the whole treebank, with just a few exceptions.

The standard dependency accuracy of the Model Two is **74.7 %**, compared with 51 % of the Model One. The following sections describe important features of the model, including their contribution to the accuracy. Section 8.13 gives a more detailed evaluation; extended evaluation results are given in Chapter 13.

We usually evaluate contribution of parser modifications to accuracy by saying what would the accuracy be just without the particular feature — as if that modification was the last one in the row. Be aware that if two or more features are turned off the drop in accuracy may be more or less than the sum of their contributions. Many features are *not mutually independent* and influence each other.

8.1 Frequency or conditional probability

The baseline model and the Model One rated dependencies according to their unconditioned probability, estimated from their relative frequency. Since the frequency was relative to the total number of dependencies in training data, all counts were simply divided by the same constant and most of the comparisons[15] could have been done using just the absolute frequencies.

There is another possible approach to ranking the dependencies. We can condition the probabilities with the depending node, i.e., the absolute frequency of the dependency will be divided by the frequency of the dependent. No wonder that such conception gives different results. If absolute frequencies were used instead of conditional probabilities, the accuracy would drop from 74.7 to 72.0 %.

8.2 Component-based building

The Model Two uses a new algorithm for the tree building. Recall the algorithm of Model One and of the baseline model (see Chapter 6). The tree was constructed from the root to the leaves. In each round the parser selected the most probable dependency from the set of allowable dependencies; that set constrained the resulting graphs to trees. A dependency was allowed, if its governor had already been added to the tree and the dependent had not.

The new algorithm builds the tree in a bottom-up manner by *connecting components*. In the beginning, there is a forest of *n* components, each containing exactly one word. Two components are connected with a dependency each round, and the

[15] If probabilities of two dependencies are to be compared, their (absolute) frequencies can be compared instead, with the same results. However, when comparing probabilities of trees, the dependency probabilities get multiplied and the comparison of the products need not be equivalent to the comparison of products of frequencies.

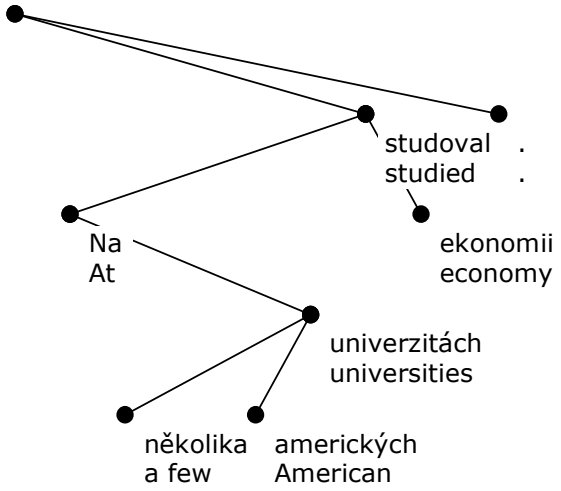


Figure 15: Na několika amerických univerzitách studoval ekonomii.
He studied economy at a few American universities.

number of components decreases by one. A dependency is allowed if its dependent is a root of a component (but not the global root) and its governor is a member of *another* component. Consider the sentence

(14) Na několika amerických univerzitách studoval ekonomii. "At a few American universities he studied economy."

There are 7 words and 49 possible dependencies.[16] Suppose the following dependency frequencies have been observed in the training data.[17]

G↓/D→	na	několika	amerických	univerzitách	studoval	ekonomii	.
#	3	0	0	0	260	31	1526
na	x	0	0	157	0	174	0
několika	0	x	1	2	0	0	0
amerických	0	0	x	0	0	0	0
univerzitách	18	20	259	x	5	1	0
studoval	57	0	0	8	x	153	0
ekonomii	36	0	1	1	11	x	0
.	0	0	0	0	0	0	x

When the old top-down algorithm was used the tree would be built in the following order:

[16] Each of the 7 words can depend on any of the 6 remaining words or directly on the root.
[17] The table contains real counts of m-tag dependencies from PDT 1.0c1.

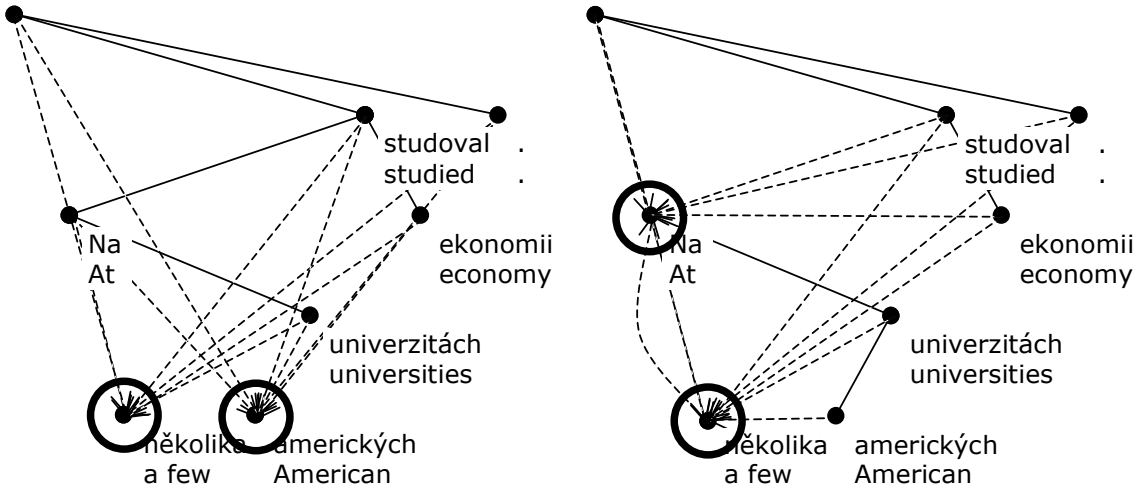


Figure 16: Allowed dependencies for top-down (left) and component-based (right) building.

Round	**Governor**	**Dependent**	**Frequency**
1.	#	.	1526
2.	#	studoval	260
3.	studoval	ekonomii	153
4.	studoval	na	57
5.	na	univerzitách	157
6.	univerzitách	amerických	259
7.	univerzitách	několika	20

The component-based algorithm changes the order to the following:

Round	**Governor**	**Dependent**	**Frequency**
1.	#	.	1526
2.	#	studoval	260
3.	univerzitách	amerických	259
4.	na	univerzitách	157
5.	studoval	ekonomii	153
6.	studoval	na	57
7.	univerzitách	několika	20

Figure 16 shows the allowed dependencies for both algorithms after the 5th round.

In our example both algorithms resulted in the same tree. Of course, this will not always be the case. For instance, using the top-down method, the dependency D(*ekonomii*, *amerických*) became allowed two rounds before D(*univerzitách*, *amerických*) did. If it was seen just 58 times in training data, it would win over its much more frequent competitor.

Zeman (1997) reports that there is no significant difference between the two approaches in the terms of accuracy: the top-down approach achieved 31 % while the component-based approach achieved 32 %. We do not repeat the comparison for Model Two because of two reasons:

- Non-trivial changes of the code would be necessary.

- The difference (if there is any) would hardly be in favor of the top-down approach. If projectivity is watched, the top-down approach will only allow a subset of dependencies allowed by the component-based approach in the same parsing state.

8.3 Searching for the best tree

The beam search of the Model One has been reimplemented in Model Two. However, the results were disappointing. The search procedure needs to employ some tree evaluation scheme, and it turned out that tree probability (product of dependency probabilities) could not serve that role. Perhaps the assumption of statistical independence of dependency probabilities was too simplistic here. Anyway, we counted that in most cases the correct tree had lower probability assigned by our model than the tree output by the parser.

	Number of trees
P(correct tree) > P(parser output)	1988
P(correct tree) = P(parser output)	1284
P(correct tree) < P(parser output)	4047
Total	7319

We would have to invent a better weighing function if we wanted the beam search to help. The tree probability alone cannot reflect all other features and constraints the parser applies when building the tree.

The final version of the Model Two does not use the beam search at a global level. It enables backtracking if the first-proposed tree meets a condition but this capability is used with moderation (see Sections 8.6.2, 8.7, and 8.12.2). We also thought about a zero filter that would try to block trees containing a dependency unknown from the training data (i.e., a dependency with zero probability). However, it would not work for the same reason the beam search did not. Unknown dependencies are simply not an important cause of errors. A simple experiment would show that among the 126030 test dependencies there were only 778 unknown dependencies in the parser output and 530 unknown dependencies in the human annotation.

8.4 Reduction of the tag set

The set of m-tags was further reduced to something very similar to the tag set used by Collins in Hajič et al. (1998) (see Chapter 1, Section Alternative Part-of-Speech Tagsets). Collins tested several reduced tag sets and reported that "the most successful alternative of this sort was a two-letter tag whose first letter was always the primary POS, and whose second letter was the case field if the primary POS was one that displays case, while otherwise the second letter was the detailed POS. (The detailed POS was used for the primary POS values D, J, V, and X; the case field was used for the other possible primary POS values.) This two-letter scheme resulted in 58 tags. Even richer tag sets that also included the person, gender, and number values were tested without yielding

any further improvement, presumably because the damage from sparse data problems outweighed the value of the additional information present."

So if the adjective *nejnepřesnější* "the least precise" had its tag reduced from AAFS4----3N---- to AAFS4----1A----, it now shrank to just A4.

The reduction scheme of the Model Two slightly differs from the one applied by Collins. Namely it continues to encode punctuation directly in its m-tag, and it converts ordinal numerals to adjectives (so, *první* "first" is annotated A1 instead of C1).

If Model-One-style reduction was used the accuracy would drop from 74.7 to 71.0 %. If no reduction scheme were applied the accuracy would further drop to 70.1 %.

8.5 Lexicalization and selective lexicalization

In Section 7.3 we discussed the surprising fact that lexicalization did not help the Model One. The same trouble has been faced with the Model Two. The following table compares accuracy for three different levels of lexicalization (λ=0 means no lexicalization, λ=1 means strictly lexical model without backing off to m-tags).

λ	**accuracy**
0	73.9
0.734375	74.7
1	54.9

Empirically we found the optimal value of λ to be 0.734375. It is doubtful whether there is a better back-off scheme that will enable better results.

We attempted to analyze the situation in the following experiment. Each time the parser decided for a dependency we searched the sentence for another governor candidate with the same m-tag. Then we had a depending node with two governor candidates: the selected candidate, and the challenging candidate. Tag-tag frequencies were the same for both candidates so if the challenging candidate should win it would have to show higher lexical frequency. The results? We found over 80000 different triples (dependent – selected candidate – challenging candidate), among which only about one tenth showed higher lexical frequency for the challenging candidate. Furthermore, only in some 200 cases this meant accuracy improvement, otherwise the challenging dependency was wrong. (The high percentage of wrong challenges may have been caused by the fact that most Model One and Model Two constraints were not reflected in the compared frequencies; but if the constraints were set in force, the number of improving challenges would not be any higher.)

Some examples of improving challenges:

Challenge	**Dep.**	**Freq.**	**Selected**	**Dep.**	**F.**	**GovTag**	**DepTag**	**Freq.**
mohly "should"	*by* <particle>	106	*bývaly* "were"	*by* <particle>	0	Vp	Vby	5540
hlediska "aspect"	*z* "from"	96	*sdružení* "association"	*z* "from"	0	N2	Rz	1108

Challenge	Dep.	Freq.	Selected	Dep.	F.	GovTag	DepTag	Freq.
měl "had"	*v* "in"	97	*představil* "introduced"	*v* "in"	9	Vp	Rv	6870
jednání "negotiation"	*s* "with"	71	*států* "states"	*s* "with"	1	N2	Rs	759
rozhodnutí "decision"	*o* "about"	51	*ředitele* "director"	*o* "about"	0	N2	Ro	807
má "has"	*právo* "right"	47	*vyhýbá* "shuns"	*právo* "right"	0	VB	N4	10577
fakt "fact"	*že* "that"	47	*většina* "majority"	*že* "that"	0	N1	Jže	396

Well, if word-on-word frequencies did not help, how about querying word-tag or tag-word? One might expect those would be less sparse and more helpful. We redid the previous experiment for both the half-lexicalized models. For the **word-tag** model (governing node lexicalized) we found 83120 triples, out of which 22839 preferred the challenging node, 812 had improvement potential (challenging dependency correct), and 368 both preferred the challenging node and that node was the correct governor.

Some examples of improving challenges:

Challenge	DepTag	Freq.	Selected	DepTag	F.	GovTag	DepTag	Freq.
má "has"	N4	1067	*vyhýbá* "shuns"	N4	0	VB	N4	10577
může "can"	Vf	790	*nepopírají* "they do not deny"	Vf	0	VB	Vf	6720
musí "must"	N1	325	*stará* "cares" or "old"	N1	6	VB	N1	21387
má "has"	P1	266	*financuje* "finances"	P1	1	VB	P1	6114
chce "wants"	Vf	255	*zvažuje* "thinks about"	Vf	0	VB	Vf	6720
ministerstva "ministry"	N2	246	*obrany* "defense"	N2	9	N2	N2	16626
zákona "law"	A2	192	*dogmatu* "dogma"	A2	0	N2	A2	38473

Beware that the cases where lexical frequencies have the most improving potential frequently involve verbs. That is because verbs often *subcategorize* for particular classes of dependents. In fact, the word-tag dependencies are parts of subcategorization frames. So even when the present results do not support the idea of

modeling the word-tag dependencies, there is still a hope that a fancier approach using subcategorization could help. We will address this issue later in Section 8.6.

The **tag-word** model (dependent node lexicalized) differs in that both dependencies (the selected and the challenging one) can be correct; nevertheless, among the 76637 triples we found, none had such property. 25204 triples preferred the challenging node, 2232 had improvement potential (challenging dependency correct, selected dependency wrong), and 839 both preferred the challenging node and that node was the correct dependent. This model also differed in that some triples occurred more than once (but this is no real sensation: there were just about two dozens of repeated triples).

The improving challenges include:

GovTag	**Challenge**	**Freq.**	**Selected**	**Freq.**	**DepTag**	**Freq.**	**Governor example**
Vp	*se* "with" or <reflexive particle>	6979	*s* "with"	1217	Rs	1514	*odcházela* "she left"
VB	*to* "it"	1361	*vy* "you"	14	P1	6114	*neznáte* "don't know"
Vp	*včera* "yesterday"	1194	*navždy* "forever"	1	Db	10026	*označil* "marked"
Vp	*který* "which"	1073	*její* "her"	0	P1	4512	*tlumočil* "conveyed"
N1	*tato* "this"	492	*která* "which"	6	P1	4586	*společnost* "company"
C=	*17*	23	*54*	5	C=	2037	*32*

A look at the last table does not promise much about the tag-word model. However it points out some frequent words whose lexical potential can help. For instance, the word *včera* "yesterday" almost invariantly depends on past-tense verbs (m-tag Vp, in contrast to VB, Vs, or Vf). For another instance, the wh-pronoun *který* "which" modifies verbs more frequently than nouns. Other pronouns like *její* "her" can be indistinguishable from wh-pronouns by their m-tag (both have P1), yet they hang on nouns very often.

Fortunately there is an alternative to the full lexicalization. It makes use of the lexical syntactic preferences of some words while not making the data sparseness problem much worse. In fact, it is a primitive usage of subcategorization. We call it **selective lexicalization.**

The lexicalized model described above took into account the lexical information for both ends of a dependency, for the governor as well as for the dependent, and it did so

for each and every pair of words in the corpus. There is evidence[18] that in many cases only one member of the dependency (usually the governor — verb, possibly also adjective or noun) is determined lexically, while the morphological class determines the other. In other cases, the dependent node has to be lexicalized as well (prepositions, subordinating conjunctions like *že* "that" etc.).

In our experiments we lexicalized all prepositions by replacing the case marker in their tag by their lemma. Thus, the R3 tag for *k* "to" (meaning that it is a preposition requiring a noun in dative) became Rk.[19] Similarly we lexicalized tags of subordinating conjunctions (J, for *že* "that" became Jže)[20], reflexive particles (P4 for *se* became Pse), pronouns, and some adverbs.

If selective lexicalization of subordinating conjunctions were turned off the accuracy would drop from 74.7 to 74.3 %. If selective lexicalization of prepositions were turned off the accuracy would drop from 74.7 to 74.5 %.

A rather complicated issue is pronouns. On one hand their original m-tags are quite complex, encode every subtle property of each word form (but not lemma), resulting in tags for just one or two words. Thus it seems appropriate to apply some tag reduction scheme. On the other hand, even before reduction the tags do not cover lemmas, which often seem to bear useful information. The simplest observation is that there are substantive and attributive pronouns. While the former (personal pronouns, e.g.) behave more like nouns, even if usually unmodified, the latter (possessive pronouns, e.g.) resemble rather adjectives. Across those categories there are wh-pronouns that are used in relative clauses and have different syntactic preferences. We tried to design new parts-of-speech, substantive pronoun and attributive pronoun, according to the sub-part-of-speech in their original m-tag. Unfortunately, the accuracy of parsing with such tags was worse than before. In contrast to that, selective lexicalization using lemmas helps.[21] For instance, the word *kterou* "which", tagged as P4 (a pronoun in accusative), would be now tagged P4který.

If selective lexicalization of pronouns were turned off the accuracy would drop from 74.7 to 74.3 %.

Adverbs are similar to pronouns in that they need to be further pruned before lexicalizing some of them. All adverbs form an open class likely to fall into data sparseness problems.[22] Some adverbs indeed have special syntactic preferences (for example, we already mentioned *včera* "yesterday" that naturally prefers past verbs over

[18] Documented in subcategorization dictionaries.
[19] The number of prepositional tags rose from 8 to 77.
[20] In fact the parser uses a more language-dependent variant, it only lexicalizes *some* subordinating conjunctions, namely *že* "that", *aby* "so that", and *zda* "whether". It would be nice to avoid such lists and lexicalize any word of the given language that is m-tagged with J,. But for Czech we found that it damaged accuracy.
[21] Slightly better results can be obtained if only some selected pronouns are lexicalized but the improvement is almost invisible and any list of Czech pronouns makes the parser more language dependent. So we kept lexicalizing all pronouns. It is possible that the "good" pronouns could be captured using the pronoun class encoded in their tags (e.g., demonstrative pronouns) but we have not investigated that.
[22] There are 2437 unique adverb forms in PDT 1.0 training data.

present verbs). In contrast, many adverbs, mostly those derived from adjectives, describe a way of doing something and are syntactically similar to each other (e.g. *syntakticky* "syntactically"). The m-tags distinguish between two types of adverbs, those able to form degrees of comparison and negation (Dg), and those that do not (Db). The latter roughly correspond to adverbs *not* derived from adjectives but unfortunately they do not include some interesting words with special behavior, like *více* "more". So we lexicalized just all adverbs that had been seen 100 or more times in the training data.[23] To give some examples for Czech: the nine most frequent adverbs are the following (all tagged Db):

Rank	Czech	English	Occurrences
1	*tak*	so	2101
2	*jak*	how	1638
3	*už*	already	1611
4	*také*	also	1566
5	*již*	already	1344
6	*ještě*	still	1314
7	*včera*	yesterday	1293
8	*tedy*	thus	990
9	*pak*	then	952

The five most frequent Dg-adverbs are the following:

Rank	Czech	English	Occurrences
10	*více*	more	946
26	*stejně*	same	448
36	*zřejmě*	perhaps	377
41	*často*	often	357
46	*méně*	less	335

The evaluation showed that such selection of adverbs is quite helpful.

If selective lexicalization of adverbs were turned off the accuracy would drop from 74.7 to 74.2 %.

We must be careful when applying the described technique to governing verbs. The number of different verbs is too big to replace their tags by their lemmas, so we cannot simply transit from the tag VB for verbs like *give* to a "lexicalized tag" Vgive — the tag dependency probabilities would lose their ability to smooth the model. Rather we sum up the lexical (Vgive) and non-lexical (VB) probabilities. This is a violation of probability laws and the resulting weight is indeed not a true probability but the real

[23] There are 140 such adverbs.

effect is positive.[24] In fact, verb-driven dependencies are honored as their relative frequency from the corpus is counted more than once. This is exactly what we would like to do with subcategorized dependencies. And verb-driven dependencies not learned from the corpus get a weight equal to their original non-lexical probability.

Selective lexicalization has also been applied to the verb *být* "to be". Its tag is always augmented with its *form* (not lemma) from the sentence, possible negative prefix and gender ending stripped. This allows the model to distinguish constructions like *bude dělat* ("he will do", *do* is governor) from constructions like *může dělat* ("he can do", *can* is governor). It is no more necessary to include "subcategorization frames" for *to be* (and it is even undesirable because this verb has too many different functions in Czech: as an auxiliary verb, as a part of a nominal predicate, as a full-meaning verb etc.)

If selective lexicalization of *být* were turned off the accuracy would drop from 74.7 to 73.2 %.

If whole selective lexicalization were turned off the accuracy would drop from 74.7 to 72.2 %.

8.6 Subcategorization

8.6.1 Introduction

Some words (especially verbs) tend to attach complements of a certain category. Such lexical syntactic preference is called **subcategorization** (or **valency**). The required modifiers of the given verb are called **arguments,** as opposed to additional — and optional — material, called **adjuncts.**[25] We call the set of arguments of a word its **subcategorization (valency) frame.**

For instance, the verb *to give* requires two noun phrases as arguments[26], the thing given and the recipient: *to give somebody something*. While some other verbs, e.g. *to sunbathe*, require no arguments (but subject) at all, they are **intransitive.** Verbs of both types can however attach optional adjuncts — such as the specification of time and place of the action (*he sunbathed yesterday in Atlantic City; he gave her the book yesterday in Atlantic City*).

Czech verbs subcategorize for part-of-speech, preposition, and morphological **case** of nouns (the latter is not present in English). Typical Czech argument classes include `N4`, `N3`, `N2`, `N7` (nouns in a particular case — 1 indicates nominative, 2 genitive, 3 dative, 4 accusative, 5 vocative, 6 local, 7 instrumental), `R`*n*(*prep*) (prepositions, *n* indicates case of the subordinated noun phrase, *prep* is the lemma of the preposition), `JS`(*conj*) (subordinated clauses headed by the *conj* conjunction — "that" would be an

[24] The effect was significant in the context of the state the parser was in when the feature was added. For the final version of Model Two, the improvement is still greater than zero but it is no more significant.

[25] The argument/adjunct (or obligatory/optional) distinction is *not equal* to the distinction between inner participants and free modifiers, as defined in FGD. See also Section 8.6.5.

[26] In contrast to common linguistic interpretation, we exclude the subject when naming arguments of a verb. Czech is a pro-drop language, which means that surface representations of sentences often miss their subjects. Therefore it is very difficult to capture subjects automatically.

example), VINF (infinitive), S (relative clause or direct speech), D (adverb). Besides verbs, some nouns, adjectives, and adverbs have subcategorization frames of their own: *zájem o něco* "interest in something" R4(o); *chudý na vápník* "poor in calcium" R4(na); *blízko něčemu* "close to something" N3.

Some example frames follow:

Frame example	**Description**	**Example**
N4	direct object in accusative	*nese zavazadlo* "he carries luggage"
N4 N3	bitransitive – accusative and dative objects	*dal bratrovi knihu* "he gave a book to his brother"
se[27] N2	reflexive particle *se* and a genitive object	*bojí se mě* "he is frightened of me"
VINF	infinitive	*musí odejít* "he has to leave"
R4(v)	PP with the preposition *v* "in" governing an NP in accusative	*věří v Boha* "he believes in God"
JS(že)	clause starting with *že* "that"	*tvrdil, že to ví* "he said he knew it"

Intuitively, a parser should profit from the knowledge of subcategorization frames, to be able to attach the right arguments to verbs. A statistical dependency model records the preferences from the point of view of the dependent. It knows which sort of governor a word wants to depend on more than on the others. A subcategorization model, on the other hand, views the problem from the perspective of the governor. It knows which sort of dependent this node *requires*, and possibly which dependent is incompatible with the others already attached, although the dependent itself wants to attach here. And even if the statistical dependency model is not lexicalized, the subcategorization is determined lexically: a subcategorization frame is a property of a *lemma*.[28]

A list of subcategorization frames can be obtained from a machine-readable subcategorization dictionary. See Chapter 9 for a discussion of how to acquire a list of subcategorization frames for a given language.

Selective lexicalization of verbs (see Section 8.5) is the first step towards using subcategorization information. It makes use of the head-lexical information; nevertheless it does not consider the context of the rest of the frame. There is still just one pair of words considered, it only differs from the other dependency in word matching: the governor is matched according to its lemma, the dependent according to its m-tag. As mentioned in Section 8.5, the best results were obtained summing up the frequency of

[27] A linguist would normally consider the reflexive particle *se* a part of the lemma of the verb: *bát se*. As we work on a level where each word has its own node, we assume a dependency between the verb lemma and the particle. Such dependency has common properties with subcategorization and is included in our frames.

[28] For further discussion why subcategorization information is important for a natural language parser, see Carroll and Minnen (1998), Carroll and Rooth (1998), and Zeman (2002).

the lemma-tag dependency with the tag-tag dependency for verbs. Although the final evaluation of Model Two revealed that the contribution of the selective lexicalization of verbs to the overall accuracy is almost invisible, it indeed has some influence on the dependents of verbs. If we evaluated just the accuracy of attaching nodes whom the manual annotation assigned one of the s-tags Sb, Obj, AuxT, Pnom, Adv (all these are typical verb dependents), we would record an improvement from 84.6 to 84.9 %.

The selective lexicalization does not need any subcategorization dictionary. If a machine-readable list of frames is available, further techniques can be explored to assist the parsing procedure. The distinction between arguments and adjuncts can be employed.

We use a dictionary generated by a system similar to the one described by Sarkar and Zeman (2000). The dictionary is a list of verb-frame pairs (together with frame frequencies) where one verb may occur in more than one pair. A frame is a list of frame members such as N4, R3(proti) etc. The order of the frame members is not significant. They are sorted alphabetically to ensure that two instances encountered with different word orders are recognized as being the same frame.

The following table shows all classes of verb children that are predominantly (i.e. more than 50 %) arguments (measured on PDT 1.0 training data, with a subcategorization dictionary automatically acquired from PDT 0.5 by Sarkar and Zeman (2000)). The dictionary contains frames for 2919 verbs. There are other 707 verbs in PDT 1.0 d-test data, not covered by the dictionary. Of the 16329 total verb occurrences in test data, 15425 (94 %) are covered by the dictionary, and 904 occurrences are not covered. The most frequent unknown verb is *bourat* "demolish", "crash" (7 occurrences).

A node was considered argument if the subcategorization dictionary contained the verb the node hung on, and there was a frame containing a member of a class matching the node. Otherwise, if the verb was covered by the dictionary, the node was considered adjunct. With unknown verbs, neither argument nor adjunct counts were changed. Note — the dictionary does not cover subjects (usually tagged N1) so the experiment rendered subjects as adjuncts.)

Class	**Total**	**As argument**	**As adjunct**	**Arg. percentage**
VINF	13094	11885	1209	90.77
JS(že)	5539	4793	746	86.53
R4(o)	2133	1734	379	82.06
N4	47756	38582	9174	80.79
R6(o)	1987	1461	526	73.53
R3(k)	3370	2131	1239	63.23
N3	8149	5131	3018	62.96
JS(zda)	185	112	73	60.54
se	15966	9355	6611	58.59
R4(mezi)	171	97	74	56.73
R4(na)	4801	2406	2395	50.11

The next table shows the most frequent classes of typical adjuncts (occurred as arguments in less than 50 %; as mentioned above, N1 is not really an adjunct, its presence in this table is caused by the method of acquiring the subcategorization dictionary).

Class	Total	As argument	As adjunct	Arg. percentage
N1	55202	854	54348	1.55
DB	32384	3327	29057	10.27
Z	23794		23794	0.00
S	18519	3392	15127	18.32
R6(v)	12759	1126	11633	8.83
N7	5747	1321	4426	22.99
N2	4522	1279	3243	28.28
R6(na)	3951	714	3237	18.07
N	2620		2620	0.00
R7(s)	3835	1522	2313	39.69
R2(z)	2932	895	2037	30.53
R2(podle)	1998	22	1976	1.10
R2(do)	3501	1733	1768	49.50
si	2718	1013	1705	37.27
R6(po)	1762	73	1689	4.14

8.6.2 Using subcategorization dictionary for parsing

Now let us pretend we have a subcategorization dictionary at our disposal. We will try to use the information contained in the dictionary to improve parsing accuracy. The following information is additional to what our dependency model already knows:

- List of arguments a verb *requires*. If the dependency model does not select a dependency between a verb and a node meeting the criteria for that verb's arguments, we can backtrack to find a parse where the argument would be attached to the verb. Nevertheless it may happen that the argument is not present in the sentence at all (due to deletions).
    - An argument cannot be attached to a more distant verb so that the dependency would skip a closer verb awaiting an argument of the same class.
- Mutual compatibility of arguments of one verb. For instance, in Czech two arguments of the same category rarely complement one verb. If so, then the subcategorization dictionary would tell, as in *učí syna*/N4 *matematiku*/N4 "he teaches the son math". Yet more generally, a verb may attract two different arguments but each in a different frame (corresponding to a different sense of the verb). On the other hand, sometimes there may be two dependents of the same kind under the same verb, one as an argument and the other as an

adjunct, as in *jedl celý týden*/N4 *kaši*/N4 "he was eating gruel the whole week".

- Probability of a frame among all frames a verb allows. This is not equal to the overall probability of a dependency.

Probably the easiest way of exploiting some of the subcat information is the following. The parser identifies all potential "subcategorized dependencies" (i.e. those that could match a verb-argument relation) and prefers them when selecting dependencies. Only if there are no allowed subcategorized dependencies, other dependencies may be selected. Unfortunately it seems that the impact of such arrangement on parsing accuracy is not significant.

A more sophisticated approach seems to be (at the first glance) to check, after parsing of a sentence has been finished, whether for each verb there is a frame in which all slots are filled. If not, look whether there is material to fill in the gaps. If so, we would try to backtrack so that the material is used.

The idea behind the backtracking device is simple. Let us just describe and remember the state of the parsing process after a new dependency has been added to the tree. During selecting new dependency to be added, let's generate all continuation states by gradually adding all dependencies allowed at the moment. One of the new states would actually become the current state in the next step while the rest would be saved for future reference.

The exponential nature of the tree-building problem poses a significant threat for such a method. Of course, blind backtracking would make the process to effectively run forever — or, more technically, to crash due to insufficient memory for new states. We should therefore return directly to a state in which a "promising material" has just been attached somewhere. (We could be more specific, e.g. we could require the word to depend directly on a particular verb, but there is no certainty that such dependency is the only valency-solving usage of the word in the given sentence. It would be quite complicated to cover all possibilities.) Even so the process does not converge and we need to limit the maximal number of backtracking returns, and the maximal number of generated states.

In our experiment we limited the maximal number of returns to 100, and the maximal number of states to 50000. Such setting is still too generous, as the parser capitulated to a 100-word sentence and crashed after having consumed enormous 2+ gigabytes of memory. On the other hand, it successfully went over the first 4419 sentences of the test set and revealed the likely achievement of the method. Out of 75145 total dependencies in this fragment, both the old and the new parser correctly assigned 55956. The subcategorization-aware version improved 56 dependencies while damaging 78. All that comes for a price of reducing the parsing speed by a factor of nearly 29 — that really is not an optimistic result.

While the slowing down was expectable, the close-to-zero (at the bad side!) impact on the accuracy deserves an investigation. It seems that no contribution of the subcategorization module is really new to the parser — in other words, the parser is probably more-or-less able to learn how to attach arguments without mastering the

argument-adjunct distinction. We will devote most of the rest of this chapter to proving or disproving that hypothesis. In particular, we will focus on the following phenomena:

- Does the parser correctly match frames when it decides whether to backtrack? (Section 8.6.3)
- How many parsing errors can be attributed to the situation that the parser attaches two mutually incompatible arguments to one verb? (Section 8.6.4)
- How many times a verb missed one of its arguments? (Section 8.6.6)
- How many times a verb got a potential argument of another verb. (Section 8.6.6)

8.6.3 Frame matching

It turns out that sometimes the parser assumes a frame slot has not been filled although it has. Such situations do not always imply poor quality of the subcategorization dictionary; they may be caused by tagging errors as well.

Examples:

*našel partnera/*N2 "he found a partner" (correct tag would be N4)

*informatika/*N1 *poskytuje údaje/*N1 "computer science provides data" (the first N1 is correct, the second tag should be N4)

8.6.4 Mutual compatibility of verb arguments

To check whether mutual compatibility of arguments is a significant cause of errors we examined the simplest case: how many times an error is or can be caused by attaching two arguments of the same class to the same verb.

Let us denote the verb modifiers (arguments or adjuncts) that do not like to see a word of the same category modifying the same verb as **jealous.** Similarly, the modifiers that do not mind will be called **tolerant.** The list of jealous modifiers may or may not coincide with the list of arguments; on the other hand, one would expect the tolerant modifiers to more-or-less coincide with adjuncts.

The following table shows the least jealous modifier classes, i.e. the words that have been repeatedly observed with one or more siblings of the same class. Please note that even if a modifier is tolerant to siblings, it is not obliged to seek one (unless it appears with a verb whose frame requires two same arguments — that is not so frequent in Czech as in English!) As a consequence, the numbers of observations of coinciding same-class modifiers are pretty small.

Class	Total	Not alone	Alone	Not alone %
Z (punctuation)	39668	18374	21294	46.32
N1 (nominative)	81844	18226	63618	22.27
DB (adverb)	39595	8715	30880	22.01
S (clause)	23377	2099	21278	8.98
R6(v)	16255	1385	14870	8.52
N4 (accusative)	35552	2360	33192	6.64
J (conjunction)	2096	137	1959	6.54

R4(v)	1212	48	1164	3.96
N2 (genitive)	5550	176	5374	3.17
JS(aniž)	88	2	86	2.27
N (caseless noun)	4594	104	4490	2.26
N7 (instrumental)	10527	213	10314	2.02
T, NX, R2(z), R6(na), R4(na), N6, R2(kolem), R2(bez), R7(s), R4(za), N3, JS(pokud), R2(od)				between 1 and 2 %

The comparatively low jealousy of nominatives (N1) that usually serve the function of subjects can be explained by two factors:

1. Two N1's often occur with the verb *být* "to be", one serving the role of subject, and the other being a nominal predicate. A very limited group of other verbs can bind nominal predicates / nominative objects as well. Out of the 18226 "not alone" observations of N1, 9788 were with *být*, and another 85 with *znamenat* "to mean" in a similar configuration.

2. Most of the other 8353 observations are caused by tagging errors. Many N1 word forms are easily confused with N4s (accusatives). Thus if two "subjects" were modifying a verb, one of them was really an object. (Of course, dual tagging errors contributed to the tolerance of N4 words, although to a significantly smaller extent.)

The dual table: the most frequent word classes that *never* appeared with a sibling of the same class.

Class	**Example**	**English**	**Occurrences**
si	*si*	himself	3020
R4(o)	*hrát o peníze*	play for money	2333
JS(protože)	*nepřišel, protože nemohl*	he didn't come because he was busy	770
R3(proti)	*bojovat proti někomu*	fight against sb.	522
JS(jako)	*pracovat jako taxikář*	work as taxi driver	448
R7(mezi)	*stát mezi dvěma domy*	stand between two houses	445
R4(přes)	*přejet přes most*	go over a bridge	339
R2(během)	*sníst oběd během pauzy*	eat lunch during a pause	324
R2(kromě)	*kromě něho*	besides him	323
R2(za)	*stát za domem*	stand behind a house	308

It turns out that jealousy has not so much to do with subcategorization. In the above table only the first two rows represent arguments to some extent. The rest are typical adjuncts.

Of course, the above observation still does not prove that avoiding same-class siblings would not improve the parsing accuracy. We ran two experiments to see whether it helps.

In the first experiment we parameterized the dependency probability with the question whether there already was another dependent of the same class attached to the same governing node. The accuracy dropped down to 64.3 %.

In the second experiment we prohibited two same-class children under one verb. Even if we were not distinguishing jealous classes from the tolerant ones, there were as few as 978 cases (in d-test data) that the parser attempted to attach two same-class words at one point, and there were only 17 cases when the parser wanted to attach three same-class words at one point. Moreover, in majority of those cases both/all three children were attached *correctly*.

If we excluded from the experiment the typical adjuncts Z (punctuation), D (adverbs), and R6 (locative prepositions), the total numbers would decrease but the cases where one of the children is attached correctly and the others by mistake would still be the rarest ones. Even if it was not so, the overall improvement potential would be so small that it hardly would pay to do anything in this field at all.

8.6.5 Free and inner modifiers

This is a good place to present another related observation of the PDT data. Besides jealousy there is another possible view of the children of verbs. **Free modifiers** can be found with almost any verb while the **inner** ones can only be found with verbs from a particular group. The largest group of that kind is the group of verbs allowing an N4 child.

There is no rule that inner modifiers must be arguments (i.e. obligatory) but in many cases the coincidence is striking. We looked for inner and free modifiers in PDT 1.0 training data.

Although the above definition of inner and free modifiers seems at the first glance to lead to a simple algorithm, in fact there are complications. If total number of verbs with which the modifier has been seen is used, less frequent modifiers will suffer as they will not get chance to show they have no preferences among the verbs. If the ratio of modifier-verb occurrences vs. modifier total occurrences is used, very frequent modifiers will suffer, as there are not enough verbs (and verbs themselves do not occur all equally likely). Therefore, we used the following heuristic:

1. Let *N* be the number of verbs a modifier has been seen with;
2. Let *W* be the total number of occurrences of a modifier;
3. Let *V* be the total number of different verbs seen;
4. Let *R* be $\frac{N}{W}$ if $W < V$, and $\frac{N}{V}$ elsewhere.

We assume that there is a threshold R_0 and that classes with $R < R_0$ are inner modifiers, the rest are free. We hope that the data is representative enough so that increasing *V* will not distort the statistics significantly. The data in the following table will help us to decide whether our assumption is plausible.

***V* = 5356**			
Class	***N***	***W***	***R***
R4(o)	172	2333	0.074

***V* = 5356**			
Class	***N***	***W***	***R***
R6(o)	174	1892	0.092
JS(že)	548	6379	0.102
VINF	553	15263	0.103
R3(k)	615	3770	0.163
N2	453	2715	0.167
N3	923	6943	0.172
…			
R4(přes)	209	339	0.617
N4	3326	39375	0.621
JS(zatímco)	187	298	0.628
…			
DB	3472	38834	0.648
…			
R2(za)	208	308	0.675
…			
R4(po)	143	198	0.722
…			
N1	3953	77233	0.738
…			
R2(místo)	114	145	0.786
…			
R3(kvůli)	113	143	0.790
…			
JS(třebaže)	10	10	1.000
R2(mimo)	14	14	1.000

The above table contains a selection of interesting modifiers; of course, full table would be much larger. Linguistic intuition suggests to place $R_0 \approx 0.625$. One must not forget that the modifiers are sorted according to surface criteria while the inner-free distinction should be bound to the meaning. For instance, R6(o) often corresponds to the meaning of the English preposition "about" (*mluvit o něčem* "to speak about st." — if used so, it is inner participant) but it can also be a time specification (*přijít o páté* "to come at five o'clock" — in this case it is a free modifier).

To summarize, the inner-free distinction can be quantified using a measure defined as *R* above. We do not attempt to incorporate it into our parser but it provides a linguistically interesting observation about PDT and, possibly, Czech.

8.6.6 Parsing errors in subcategorization

This is the most crucial part for determining whether the subcategorization dictionary could have helped the parser at all. We analyzed the output of the parser on the d-test

data and counted all errors where a node should have been attached to a verb but was attached elsewhere (even to another verb if it had a different lemma).

There were 8262 such errors. If we excluded the verb *být* "to be" (too many frames and auxiliary usages) and all the verb-child pairs not covered by the subcategorization dictionary, only 1415 errors remain (~1.5% of all errors). If we were able to perfectly eliminate all of them, the parsing accuracy would rise by approximately 1%; however, at the same time the subcategorization procedure would have not to introduce any new error. Unfortunately, that is not our case.

The following table lists the most frequent subcategorization errors.

Erroneously attached node	Verb missing that node	English translation	Number of cases
S	*říci*	tell	40
N4	*mít*	have	29
N2	*dosáhnout*	reach	16
VINF	*muset*	must	15
S	*říkat*	say	14
N4	*získat*	acquire	13
N2	*mít*	have	13
S	*vědět*	know	13
VINF	*mít*	have to, shall	11
S	*uvést*	state	11
se	*stát-2*	become, happen	10
N7	*stát-2*	become	10

The most frequently misplaced subtree is the direct speech of saying verbs; in those cases, the "thief" is the tree root #:

Erroneously attached node	Verb missing that node	Word where the parser attached it	Number of cases
S	*říci* "tell"	#	30
S	*říkat* "say"	#	10
S	*uvést* "state"	#	10

Often (695 cases) the "thieves" are modal or similar verbs and the "victims" are infinitives:

Verb missing a node	Word where the parser attached it	Number of cases
stát-2 "become"	*mít* "shall"	6
stát-2 "become"	*moci* "can"	5
účastnit "attend"	*mít* "shall"	4

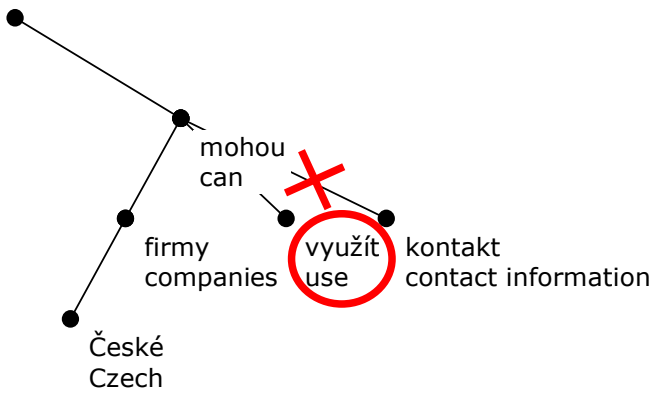


Figure 17: Prohibited skipping of an infinite verb.

vyjádřit "express"	*odmítnout* "refuse"	4
uložit "deposit"	*moci* "can"	4

Sometimes it is impossible to correct such situations without making the tree non-projective (see Sections 7.1 and 12). The rest can be solved by a simple heuristic, without any subcategorization vehicle (see Section 8.6.7).

Finally let us note that we do not evaluate the errors from the opposite point of view — whether there is a node attached to a verb, despite it does not belong to the verb's subcategorization frame. We would have to filter out all nodes but "typical arguments". As we showed in the subsections above, it is difficult to recognize typical arguments by any metric proposed, and there is never a sharp delimiting line.

8.6.7 No skipping of potential parent verbs

If the parser selects a wrong longer dependency and the correct governing node lies between the proposed governor and the dependent, we say that the dependency is **skipping** its correct parent in favor of another node. Since many errors of this kind are related to subcategorization of verbs, we are going to discuss the matter here.

For an instance of skipping dependency, consider the following example:

(15) *České firmy mohou využít kontakt…* (lu01:49)

"Czech companies can use the contact information…"

We have examined the verb skipping in the output of the parser on the d-test data. Of course not every skipped verb corresponds to an error. The following table shows the most frequent erroneous cases. The fourth column shows how many times a skipping of some type was wrong; the fifth column shows the share of these cases on the total occurrences of skipping of that type.

Skipped node	**In favor of**	**Dependent**	**No of errors**	**Error %**
Vf (infinitive)	VB (present)	N4 (accusative)	168	91.3
Vf (infinitive)	Vp (past)	N4 (accusative)	120	90.9
Vf (infinitive)	Vp (past)	Z, (punctuation)	98	89.1

Skipped node	In favor of	Dependent	No of errors	Error %
Vf (infinitive)	VB (present)	Z, (punctuation)	92	82.9
Vp (past)	# (root)	J^ (coord)	88	40.2
Vf (infinitive)	VB (present)	J^ (coord)	87	92.6

Most useful for the parser are the skipping patterns where the skipping dependency is consistently wrong. If we placed the threshold of "consistently" to 90%, the total number of occurrences of such patterns would be 4946.

If we did not mind the tag of the dependent node, it would become even more apparent that infinitives prevailed among the omitted (skipped) nodes. On the other hand, the persuasiveness (last column) would drop down.

Skipped node	In favor of	No of errors	Error %
Vf (infinitive)	VB (present)	852	89
Vf (infinitive)	Vp (past)	667	87

We can make the parser to learn from its errors by supplying it the list of tag triples (skipped-governor-dependent) that have been observed at least once and at least 90% of the observations were errors. Unlike in the above investigation, the parser must not learn that on its output on the test data. We have to train the parser on a subset of the training data and to learn its tendency to skipping errors on another subset (held-out data). We used the PDT 1.0 mtrain and mtest data (see Chapter 5) in place of these subsets. We filtered out everything but triples seen at least 5 times and wrong in 90% of the cases. The experiments further revealed that only learned skippings of verbs contributed to better parsing accuracy. Skipping of other nodes could not be successfully learned from data but see the Sections 8.12.6 and 8.12.7 for some hard constraints of this sort.

After all filtering was done 199 tag triples survived.

The parser was re-trained on the whole training set before applying it to the d-test data, using the above blacklist. If the blacklist were not used the accuracy would be 74.5 instead of 74.7 %.

8.7 Fertility

The Model Two proposed so far does not employ any supporting fertility model as described in Section 7.4. In the present section we first discuss fertility-related errors in the output of the Model Two parser, then propose some approaches to avoiding the errors, and evaluate them.

We set up the following constraints to classify a parsing error as fertility-related:

- A node has a number of children different from the number in the correct parse.
- The correct number of children is equal to the typical number of children of a node with the given m-tag, as defined by the fertility model.

- A particular number of children is considered typical for a particular m-tag if and only if:
  - The m-tag occurred at least 100× in the training data.
  - It occurred with the typical number of children in at least 80% cases.

Note that even if the constraints were not defined the total number of fertility errors would not be equal to the total number of wrong dependencies. A fertility error is proper to a governing node and can account for several incorrect dependencies.

We found 2111 fertility-related errors in the output of the Model Two parser. Out of that, 1164× a typical leaf had one or more children, 342× the node should have (but had not) exactly one child, and 605× the node should have (but had not) exactly two children. No m-tag has typical fertility of three or more, according to the above constraints.

The following table shows some examples of fertility-related parsing errors. It is no general overview of fertility preferences. Cases that are not interesting from the point of view of the parsing error analysis are not included.

Occurrences	**Tag**	**Example**	**Proposed number of children**	**Correct number of children**	**Probability of the number of children (%)**
198	A1	*etnické* "ethnic"	1	0	81.7
98	A2	*identického* "identical"	1	0	91.8
56	TT	*ano* "yes"	1	0	84.9
56	Dg	*mírně* "moderately"	1	0	84.2
47	Pse	*se* "oneself"	1	0	98.9
1164	**Subtotal 0**			**0**	**≥80.0**
46	Rna-1	*na* "on"	2	1	96.7
38	Rv-1	*v* "in"	2	1	95.4
27	Ro-1	*o* "about"	2	1	99.0
19	Rs-1	*s* "with"	2	1	88.6
15	Rpro-1	*pro* "for"	2	1	98.6
342	**Subtotal 1**			**1**	**≥80.0**
173	#	<root>	1	2	86.1
148	Jže	*že* "that"	3	2	87.8
83	Jže	*že* "that"	4	2	87.8
54	Jže	*že* "that"	1	2	87.8
29	#	<root>	3	2	86.1
605	**Subtotal 2**			**2**	**≥80.0**
2111	**TOTAL**			**0 or 1 or 2**	**≥80.0**

The largest group is that of typical leaves. It includes particles (TT), interjections (II), many kinds of pronouns (P), adjectives (A), adverbs (D) etc. Expectedly, typical one-child nodes are prepositions; the errors included both no child and two children, although the latter were more frequent. The sentence root has typically two children: the main verb and the final punctuation mark. Most other typical two-children nodes are subordinative conjunctions such as *že* "that", *aby* "so that", *zda* "if"… In these cases, one child is the predicate of the subordinated clause; the other is the comma before the conjunction.

There are two separate problems that cannot be solved the same way: **too few children** and **too many children**.

If a node has reached its quota of children, this will be realized immediately and attaching new dependents can be stopped or penalized. (Of course we cannot be sure that the right children have been attached first, or that there is still a better place for attaching the new candidates.)

If a node lacks children, this will be realized after the analysis has been completed, and that is quite late. We could backtrack to a back-up state but the process would suffer from the same drawbacks as the subcategorization backtracking system (cf. Section 8.6.2). What remains is to give better weights to dependencies satisfying unsatisfied nodes during the tree-building process.

We have developed and tested three modifications of the fertility model, described below. Unfortunately none of them has a positive influence on the parsing accuracy. This result contradicts the one in Section 7.4 for Model One. What happened since then? We believe that there is a general answer applicable not only to the fertility models but also to other unsuccessful "improvements". The model has become rather complex and any new modification can produce undesirable side effects.

8.7.1 Full fertility model (FFM)

Each m-tag has a fertility distribution (four probabilities: that of 0 children, 1 child, 2 children, and 3 or more children). When considering a new dependency governed by a node with that tag, we compute the probability that the node will have more children than it currently has. We then multiply the probability of the dependency by this probability.

Example 1: a node tagged Jže currently has 1 child. The fertility distribution of Jže is 87.8 % two children, 6.9 % one child, 5.2 % three or more children, and 0.1 % zero children. The probability that the node will have more than one child is (87.8 + 5.2) / (87.8 + 6.9 + 5.2) = 93.1 %. We omit the count for zero children because at this point we are sure that this node will not have zero children (unless some backtracking is applied).

Example 2: a node tagged Z, (comma) currently has 1 child. Commas display strong preference for zero and some preference for two children, but they almost never end up with one child. So if ever the zero/one border is broken, we must finish the process and find the other child. The fertility distribution for Z, is: 86.330 % no child, 7.794 % two children, 5.871 % three or more, 0.004 % one child. The probability that

the node will have more children than the one it currently has is (7.794 + 5.871) / (7.794 + 5.871 + 0.004) = 99.971 %.

Evaluation: FFM depressed the overall accuracy from 74.7 % to 74.5 %.[29]

8.7.2 Typical fertility (TFM)

Only nodes with strong preference (≥ 80 %) for one particular number of children are affected. If the node shows preference for its current number of children (or even for a smaller number), it will get a fertility weight of 0. If on the other hand it needs one or more additional children, it will get 1. All nodes whose preferences are not strong will get a weight of 0.5.

Evaluation: TFM depressed the overall accuracy from 74.7 % to 74.0 %.

8.7.3 Checking fertility quota (QFM)

Similar to TFM but only exceeding the child quota is being checked. Nodes that have reached or exceeded the quota will get zero. All other nodes including those without strong preference will get one.

Evaluation: QFM depressed the overall accuracy from 74.7 % to 74.0 %.

8.8 Governor-dependent distance

The Model One recorded whether the governor is adjacent to the dependent or not (cf. Section 7.5). The Model Two adds a third state to this feature: whether or not there was a comma in between. If this feature is turned off the accuracy will drop to 72.9; interestingly enough, if we turn off even the adjacency parameter, the accuracy further drops to 72.4 only, although for Model One this feature used to have much more importance. We also experimented with recording the number of interfering commas but it did not bring any further improvement.

Some authors of publicly unreleased parsers (see Chapters 14 and 15) suggest that dividing weight (probability) of each dependency by the distance between the two nodes helps. We found it almost useless: it improves the accuracy *very little* but at least it does not hurt.

The following table shows statistics of dependency distance in the PDT 1.0 training data. The longest dependency encountered is equal to the maximum length of a sentence: it is the dependency between the root and the final punctuation mark. The longest dependency on a node other than the root is 163 words long.

Distance	Occurrences	Percentage
1	590884	47.1
2	236788	18.9
3	112236	8.9

[29] A table of the most fertile m-tags would be led by those most often rooting a coordination: `Z,` (maximum of 85 children encountered in PDT 1.0 training data), and `J^`. Record number of children for the sentence root is 28. A verb (`Vp`) with 19 children would win the contest among full-meaning words.

Distance	Occurrences	Percentage
4	69993	5.6
5	45459	3.6
…		
194	1	0.0

A related problem is skipping closer governors in favor of more distant ones — see also the Sections 8.6.7, 8.12.6, and 8.12.7.

8.9 Coordinations

We introduced *coordination* in Section 4.2 as a linguistic relation that is described using (usually) more than one dependency.[30] Since dependency structures are better suited to describe subordination, dealing with coordinations requires some extra care in this environment.

The major and most problematic difference between coordinations and subordinations can be described simply. Subordination is a vertical relation between a superior and a subordinated word. Such relation can be perfectly modeled by exactly one dependency where the parent is the superior and the child is the subordinated word. Coordination, in contrast, corresponds to two or more dependencies, and really important is the relation between the children. No parser can succeed in rendering coordinations without looking at more than one dependency at a time — and the Model Two described so far does not so.

[30] Recall that our *coordination* corresponds to both *coordination* and *apposition* in the sense of PDT.

To see the problem, consider the following sentence.

(16) Jedním z míst, kde mohou získat komplexní informace z České republiky i ze zahraničí, je Hospodářská komora ČR. "One of the places where they can get complex information from the Czech Republic as well as from abroad is the Business Chamber of CR."

The treebank structure of sentence (16) is shown in Figure 18. A structure generated by a coordination-unaware parser is shown in Figure 19. There is one coordination governed by the conjunction *i* "as well as". Coordinations are frequent and most coordinative conjunctions have been seen with coordination members of various

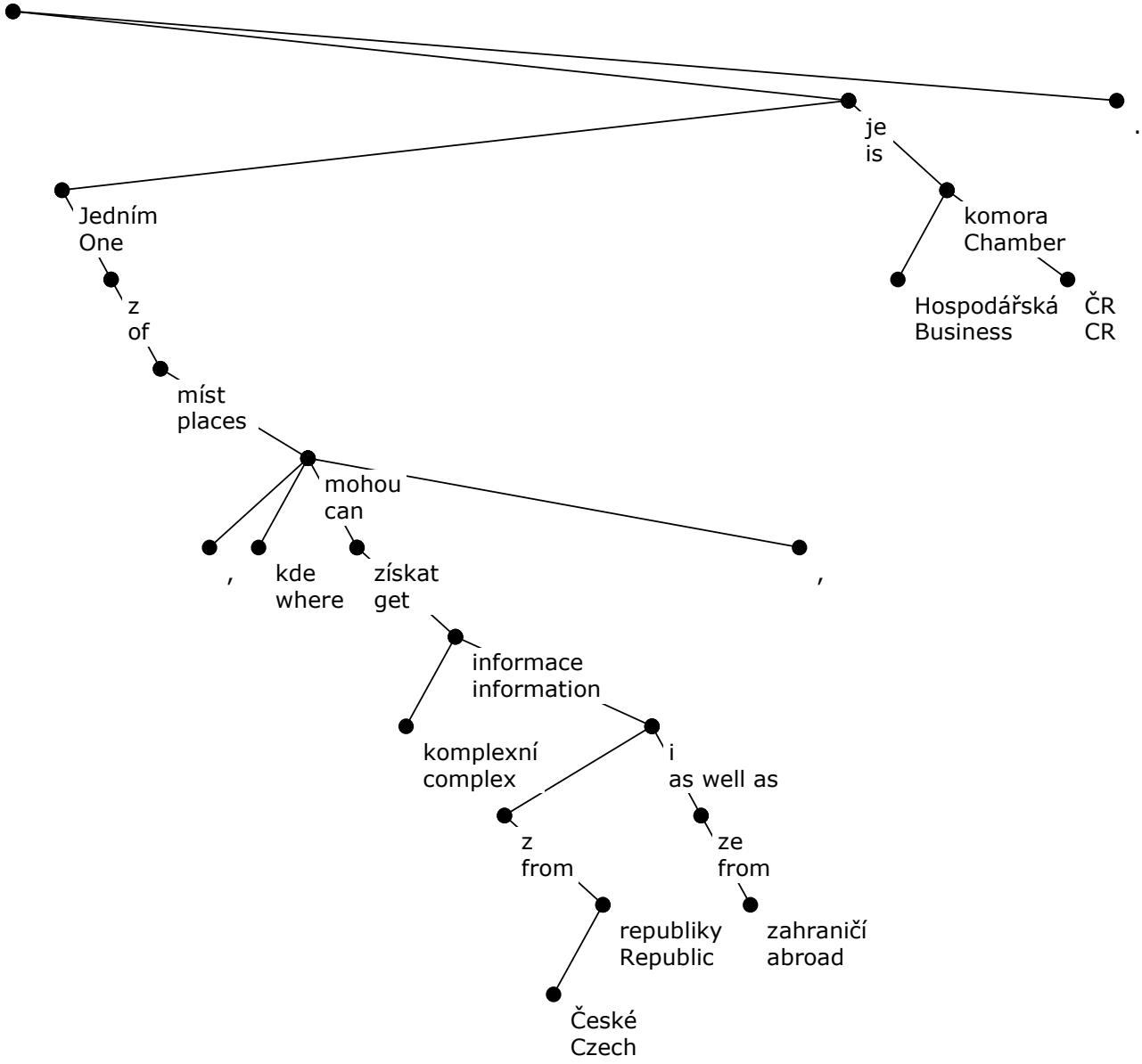


Figure 18: Jedním z míst, kde mohou získat komplexní informace z České republiky i ze zahraničí, je Hospodářská komora ČR.
One of the places where they can get complex information from the Czech Republic as well as from abroad is the Business Chamber of CR.
[correct]

types. However, all members of a coordination always agree in type![31] That is what the parser has not learned. It eventually attaches anything it finds to a conjunction, as the conjunction is known to have governed many different coordinations, with members of different type. The conjunction *i* in sentence (16) in Figure 19 is a good example of such "magnetic" node. It incorrectly attracted a numeral (*jedním*), a verb (*mohou*), and a preposition (*ze*).

There is yet another problem with the representation of coordinations. Suppose a coordination of nouns is to fill a slot in a verb's subcategorization frame. The nouns agree in case with each other, and their case is the one required by the verb. But all that is invisible from above! It is always the root of a subtree what represents it in the eyes of its governor. Here the conjunction is. No trace of the case of the nouns, even no trace that they are nouns! The verb would be foolish to accept such structure, and if it does, it risks getting a coordination of unexpected things, say, infinitives.

One possible solution is to transform coordinations into a form more suitable for parsing. We have explored several transformations. An example: the head of the coordination is its rightmost member, any other member is attached to its right neighboring member; the conjunction and the commas are attached to the closest

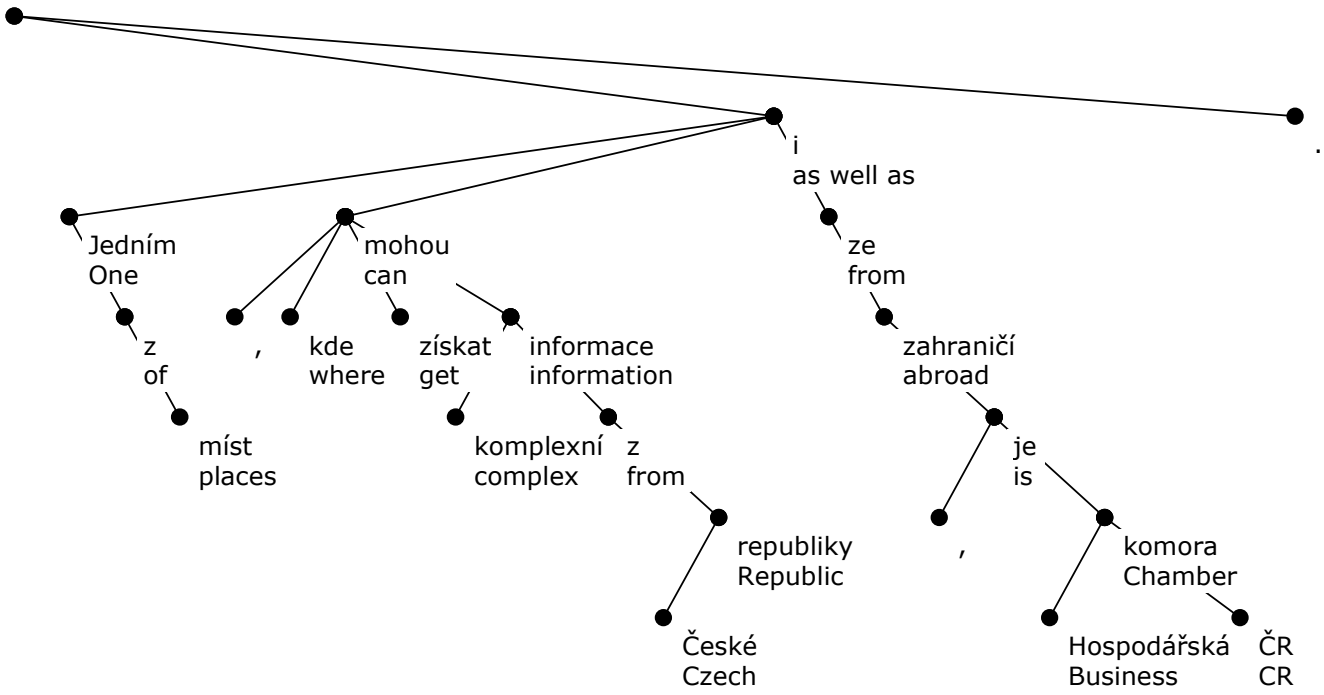


Figure 19: Jedním z míst, kde mohou získat komplexní informace z České republiky i ze zahraničí, je Hospodářská komora ČR.
One of the places where they can get complex information from the Czech Republic as well as from abroad is the Business Chamber of CR.
[wrong]

[31] We do not give any classification of coordination types. Roughly speaking, phrases of the same type share the part of speech, nouns, adjectives, pronouns and numerals agree in case. Prepositions need not be identical but they should belong to the same semantic class (*v domě a na silnici* "in the house and on the road"); the same holds for adverbs. Verbs need not agree in tense and other morphological categories.

member to their right. If a transformation of that sort could have been pre-applied to both the training and the test data, the parser would improve in coordinations. However, we would be using different (transformed) test data than other researchers use, and the results would be incomparable. So we would have to invert the transformation after parsing finished, but then we would face additional troubles (the main one: how to decide that a chain of nodes of similar type is a coordination).

We finally used another approach. In the learning phase we use s-tags to recognize coordinations. Inside a coordination we do not record dependencies. Instead, we record all pairs of m-tags that appeared as members of the same coordination. This way the parser learns to recognize words of the same coordinative type. There is even some level of robustness, crucial for dealing with tagging errors. If the tagger wrongly assigns case to one noun and correctly to the other, the training module remembers the pair. Later the parser will know that N2 is most likely to join another N2, but a coordination of N2 and N4 is not impossible either. If the same tagging error occurred in test data, a rule-based parser would discard such pair due to non-matching types, but a statistical machine has still a chance.

We further record a probability that a word is coordinative conjunction (we do not trust that the m-tag is J^, as there are other words capable of serving the same function, e.g. the comma).

The parsing module has been modified as well. So far it just compared frequencies of allowed dependencies. Now it also looks at allowed pairs of coordination members. A pair is allowed if there is a word in between that could serve as the conjunction and the dependencies of both members on that word are allowed. The conjunction must not be involved in any other coordination yet. A pair is also allowed, if its right member is already a member of a coordination, and its left member would enlarge that coordination. In that case the conjunction is not between the two members but somewhere right of the right member; a comma is between the two. The dependency of the new member and the comma on the conjunction must be allowed.

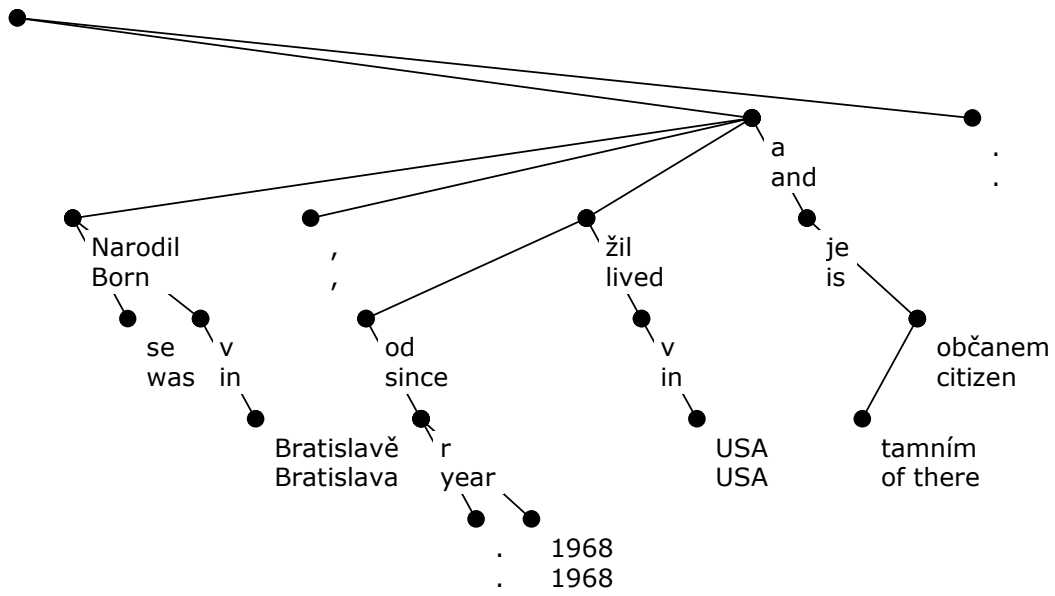


Figure 20: Narodil se v Bratislavě, od r. 1968 žil v USA a je tamním občanem.
He was born in Bratislava, since 1968 he has been living in USA, of which he is citizen.

If a pair is allowed, its probability is multiplied with the probability of the conjunction to be a conjunction, and compared to the probabilities of other coordination pairs and normal (subordinative) dependencies. It is not clear to which extent the probabilities of pairs are comparable to the dependency probabilities. We were not able to find a solid theoretical grounding to put both measures into one model. Empirically we found that favoring the coordinations improves parsing accuracy. If the learning module recorded each coordination three times, the accuracy was better than for the true counts. Anyway, even without that intervention the new processing of coordinations helped the parser a lot. If it is excluded from the final version the accuracy will drop from 74.7 to 73.2.

It turns out that verbs do not hesitate to coordinate even if they do not agree in tense. Thus it is correct to say

(17) *Narodil se v Bratislavě, od r. 1968 žil v USA a je tamním občanem.* "He was born in Bratislava, since 1968 he has been living in USA, of which he is citizen."

The coordinated verbs include *narodil* "was born" and *žil* "lived" (past tense), and *je* "is" (present tense). This led us to slightly change the m-tag reduction scheme. Coordination events are recorded with only two kinds of verbs, infinitives (Vf), and finite verbs (VB). Normal dependencies (subordinations) keep the default reduction scheme described in Section 8.4. The change of the scheme improved accuracy.

It remains to describe how the parser learns and creates subordinative relations between a coordination and its governors or dependents. The parser assumes that a coordination **inherits** the morphological properties from its members. Thus the learning module steps through the members, collects their m-tags and replaces the m-tag of the root (usually J^ — coordinative conjunction) with the set of inherited m-tags. If one or

more of the members are nested coordinations, the procedure steps down recursively.[32] The set can be dealt with as if the tags came from non-disambiguated MA. A rather surprising result however is that the accuracy is better if we take just the m-tag of the first or the last member as the representative.

An analysis of experimental results revealed one more surprising detail. Although PDT apposition seems to share syntactic properties with PDT coordination, covering both with the same procedure has not proved as improving accuracy of the parser. Better results are sometimes obtained when the parser is ordered to deal with dependencies building an apposition the same way as with any subordinative dependencies. Anyway the difference is small and statistically insignificant.

8.10 Special treatment of short sentences

Some sentences, especially short ones, have non-standard contents and structure. PDT consists mostly of newswire text and there are special "sentences" in virtually every article, usually at its beginning or end. A frequent example is the pattern

(18) Brno (jak) – "Brno (jak) –"

where the first word is a city or location, and the token in parentheses is a signature of the author of the article. The PDT structure of such sentences is simple and falls into the umbrella class (see Chapter 6, Figure 21): 0,0,0,0,0. However, the parser tends to build structures 2,0,2,2,0 it learned from appositions (Figure 22).

The author's signature, usually a two-to-three-lowercase-letter string derived from author's name, may cause additional trouble as it often coincides with a normal word. So, *jak* is also the adverb "how" or a "yak" (Himalayan cattle); *tom* is the local case of the pronoun *to* "the"; *top* can be classified as imperative of *topit* "to drown"; *do* "into" and *ad* "ad" (from Latin) are prepositions; and *hop* can be a particle or an interjection ("whoops").

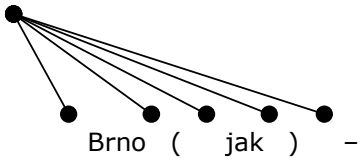


Figure 21: The correct parse of the sentence (18).

[32] Note that further technical complications appear if the members are prepositional phrases or relative clauses. A coordination member is recognized by its s-tag ending with _Co or _Ap. But a preposition always has the s-tag AuxP (not AuxP_Co) and first the word depending on the preposition bears the sign of coordination membership. Similarly a subordinated clause is headed by a conjunction such as *že* "that", and the conjunctions are invariantly s-tagged as AuxC. First their dependents (usually verbs) have some useful s-tag, e.g. Adv_Co. A reader knowing all these peculiarities from behind the scene might expect we collect also m-tags from the deeper level. But it is not so. We want to capture the coordination type, e.g. the semantic class of the prepositions involved. So we have to look for children's s-tags to see whether this is a coordination member but then we take the parents' m-tags to represent the member.

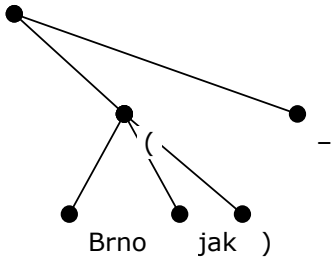


Figure 22: Parser-suggested structure of the sentence (18).

The constant length of such sentences, the context of the parentheses and the dash at the end could help recognizing situations where the structure from Figure 21 should be used. Unfortunately the Model Two as described so far is not able of capturing such contexts.

See the accuracies of sentences of particular lengths in Figure 23. There is a remarkable gap for 5 words, mostly caused by sentences of the class just described.

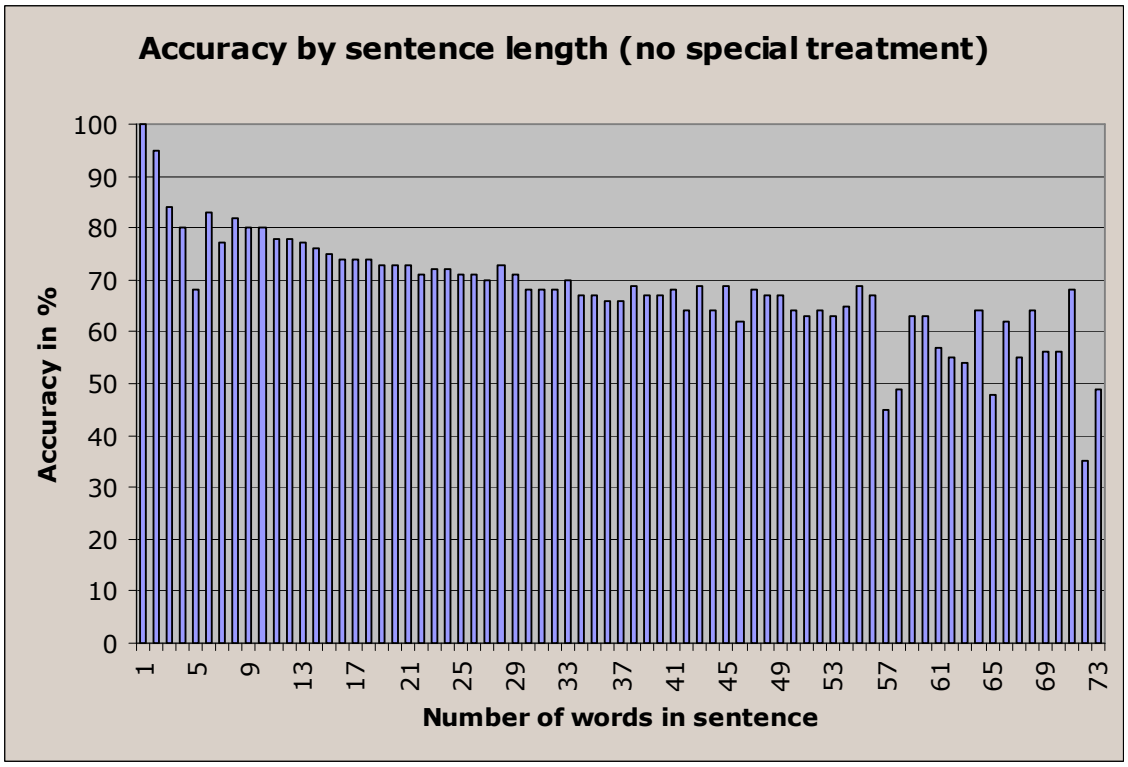


Figure 23: Accuracy by sentence length.

Less visible but still influential are similar special sentences of other lengths.

As a solution we tried to handle each short sentence (up to eight words) in one piece. We learned **morphological patterns** of short sentences from the training data. A

morphological pattern of a sentence is the sequence of m-tags of the words in the sentence. For each pattern we learned the most probable dependency structure. If a sentence with an unknown pattern was to parse or if there were competing structures for that pattern without a convincing winner[33], we parsed the sentence classically word-by-word. The accuracy improvement for short sentences is shown in Figure 24.

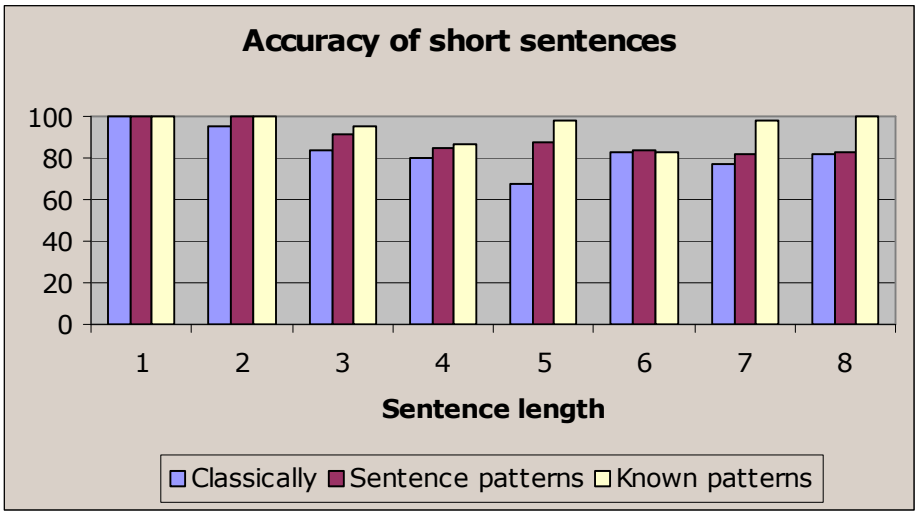


Figure 24: Accuracy comparison of short sentences parsed classically or according to their pattern. The third column gives a separate evaluation of the sentences with known patterns and thus it makes no use of the classical approach even as back-off.

Actually the requirement of "convincing winners" did not change anything because every known pattern showed clear preference for one structure. It also did not help to consider patterns seen only once unknown — accuracy dropped for each of the eight lengths by approximately 1 %. The accuracy of the pattern approach is very high and there is not much space for further improvement. The space shown in Figure 24 corresponds almost entirely to unknown patterns, i.e. sentences that had to be parsed classically despite of their length.

Note the third column for the lengths of 4 and 6 in Figure 24. Here even the known patterns do not perform as well as usual. It is because of inconsistencies in PDT 1.0. Unfortunately some sentence patterns were so special that even the human annotators were not able to assign the same structure for all instances of virtually the same sentences. It is also possible that the particular annotation guideline changed over time.

For example, there is the four-token sentence

(19) Vladimír Mišauer, Bratislava (<first-name> <last-name>, <location>)

[33] A structure is a convincing winner if its frequency is at least half of the total number of occurrences of its pattern.

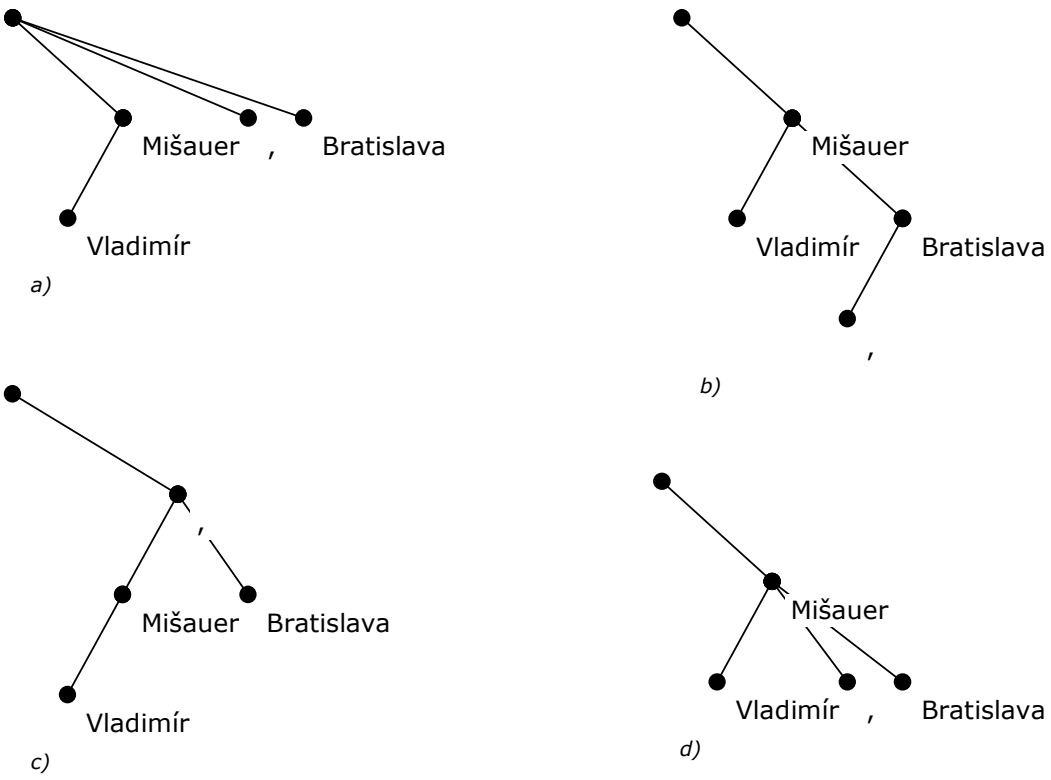


Figure 25: Four different structures of the pattern <first-name> <last-name>, <location>*, all found in PDT.*

The PDT 1.0 training data give three (!) different annotations for this sentence: 35× the 2,0,0,0 structure (Figure 25a), 12× 2,0,4,2 (Figure 25b), and 8× 2,3,0,3 (Figure 25c). The bad news is that in test data of the same corpus, it is 2,3,0,3 what wins (15 occurrences), plus there are two instances of yet another opinion, 2,0,2,2 (Figure 25d). No wonder that a statistical model is unsuccessful if training data have been annotated observing different rules than those for test data!

The same reason probably caused the sub-optimal performance of the six-word patterns (actually the problematic four-word pattern is a subset of the most frequent six-word pattern).

Since there is no statistical way to render such sentences correctly, the only help would be a hard-coded rule bound on the problematic pattern.

The widest-spread inconsistent pattern is 5 words long; however, it does not pose so serious threat because it has the same preferences in training as in test data. It is the sentence

(20) Foto Martin Přibyl – LN "Photo by Martin Přibyl – LN[34]"

[34] "LN" is abbreviation of *Lidové noviny*, a newspaper's name.

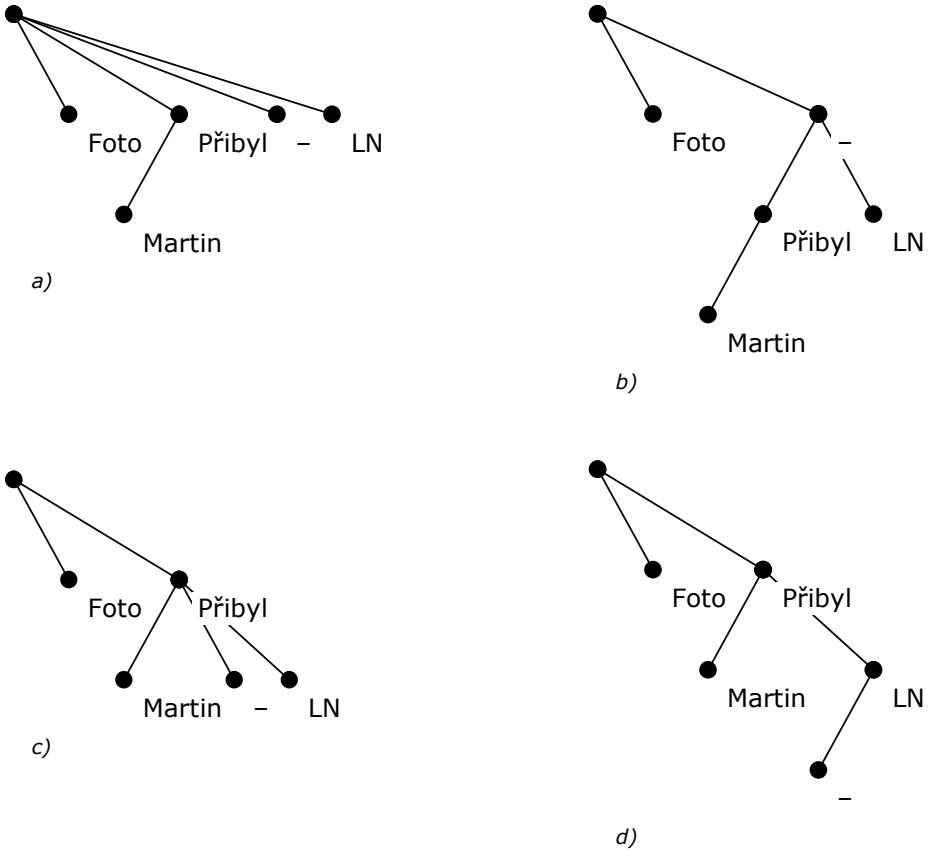


Figure 26: Four different structures of the pattern Photo <first-name> <last-name> – <agency>*, all found in PDT.*

We figured out that this sentence has been annotated 23× as in Figure 26a (0,3,0,0,0), 6× as in Figure 26b (0,3,4,0,4), 2× as in Figure 26c (0,3,0,3,3), and 2× as in Figure 26d (0,3,0,5,3).

If all short sentences are processed classical way the accuracy will drop from 74.7 to 74.5 %. A survey of the final accuracy on short vs. long sentences is presented in Chapter 13.

8.11 N-tuple patterns

In analogy to the special treatment of short sentences (see the Section 8.10), we have tried to learn structures for patterns of m-tag n-tuples. If there was an *(uninterrupted)* sequence of m-tags, for which it held that:

- the sequence (pattern) occurred at least five times in the training data,
- in at least 90 % of the occurrences the training data suggested the same structure for the pattern, and
- the suggested structure was continuous, i.e. it had one and only one root node,

we would insert the structure as a subtree of the dependency structure being built. The n-tuple-pattern model was applied before the core Model Two was, and the

core could influence only the nodes left untouched by the n-tuple model. N-tuples for $N \in [2 \ldots 10]$ are modeled.

The accuracy drops to 73.8 % when n-tuple modeling is turned off. One might expect an even more significant accuracy improvement. Note however that the requirement that the n-tuple member nodes follow one after the other is quite strong and that it will mainly cover trivial structures that the parser can learn anyway. We think that a more promising approach would be to model n-tuples of adjacent *exposed headwords*,[35] i.e. roots of subtrees already constructed. In fact, such approach would be close to data-oriented parsing (DOP; see Bod et al. (2003)).

8.12 Hard constraints

Finally we included some rules that have nothing to do with probabilistic modeling. The rules have been designed by a human, with knowledge of the language (Czech) and of the PDT annotation scheme. They are highly language- and domain-dependent, which has to be borne in mind should the parser be applied to different data. Because the statistical model is not allowed to interfere with the rules, we call the rules **hard constraints.**

Including the hard constraints is useful to document phenomena that the statistics cannot capture to a desirable extent. After all we want to build as good a Czech parser as possible and this is just one of the steps towards that goal.

8.12.1 Attaching of the final punctuation

If the last word in sentence is tagged as punctuation (its original m-tag is `Z:----------`), attach it to the root (by assigning $d(w_n)=0$). This is the first step in building the tree.

If the constraint is turned off the accuracy will drop to 72.5 %.

8.12.2 Children of the root

Partition the dependencies of something on the root into two classes, according to the presence or non-presence of at least one verb in the sentence. Technically the verb flag replaces the flag for dependency direction (left/right) because every dependency governed by the root goes to the right.

If the sentence contains a verb, it is more likely that the root will have only two children — the final punctuation, and a verb or a coordination of verbs. If there is no verb, it is probably an elliptical sentence where several words can be attached directly to the root.

The main weakness of this rule appears when the sentence is a heading containing a relative clause, such as

(21) Muž, který se bál hromu "A man who was frightened of thunder"

— the head of such sentence and the only child of the root would be the noun "man" while the verb "was" would depend on "man".

If the constraint is turned off the accuracy will drop to 74.6 %.

[35] This term was introduced by Chelba and Jelinek (1998) for their *structured language model*.

8.12.3 Root fertility

Even when the rule 8.12.2 is in effect the root is still too magnetic and in many sentences it attracts more children than desirable. Then it is possible to apply a rule that prohibits any root to have more than two children. It is a rather strong and potentially hurting constraint because there indeed are correct trees with more fertile roots. Nevertheless the experiments showed that the Model Two achieves better accuracy if such a rule is in effect than if it is not.

The rule is algorithmically enforced the following way. After completing the tree a procedure checks the number of the children of the root. If it exceeds 2, the probabilities of the dependencies are proven. The final punctuation and the most probable other child survive, all the others are detached and re-attached to another place; the search for the substitute attachment works similarly as the search for the most probable dependency during the main phase of parsing. Weakness: this approach does not allow the rest of the tree to reflect the change. More specifically, it is not possible to replace a single verb with a coordination of verbs, of which the original verb would be member.

If the constraint is turned off the accuracy will drop from 74.7 to 74.4 %.

8.12.4 Inter-comma segments

This rule originates in the observation that commas delimit autonomous segments of the sentences. Most often a segment between two commas forms a complete subtree and there is only one dependency skipping its borders — the one connecting the subtree root with an external governor.[36]

Therefore we introduced a hard constraint that blocks all comma-skipping dependencies before the whole segment is connected into one tree component. For the purpose of this rule the sentence borders are equivalent to commas (they also delimit an inter-comma segment). The artificial sentence root is viewed as forming an extra segment that is simultaneously the left neighbor of the leftmost segment of the sentence, and the right neighbor of the rightmost segment of the sentence. It effectively makes from the sentence a circle. Thus the dependency of the final punctuation on the root can be added as the first arc in the tree and it still does not violate the inter-comma segment constraint.

[36] Often an inter-comma segment corresponds to a subordinated clause. See Chapter 12 of Kuboň (2001) for related research on clausal boundaries.

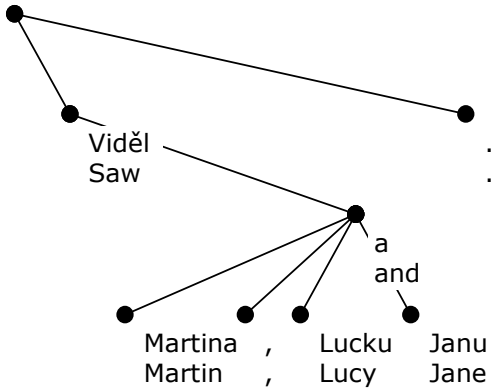


Figure 27: Viděl Martina, Lucku a Janu.
He saw Martin, Lucy, and Jane.
[counterexample]

The rule is not 100 % true. A counterexample is the sentence

(22) Viděl Martina, Lucku a Janu. "He saw Martin, Lucy, and Jane."

Although "Martin" lies in the same segment as "he saw", it must first enter the coordination with "Lucy" and "Jane", and than the whole coordination connects to the verb. An unexplained surprise is that if we exclude coordinations from the inter-comma rule the accuracy falls off. Nevertheless the overall contribution of this rule is positive. If it is turned off the accuracy will drop from 74.7 to 74.2 %.

8.12.5 Nothing may hang on a comma

A comma can govern a dependency only as a technical root of a coordination. After the parsing of coordinations has been reserved to a separate module of the parser, the subordinative dependencies should never be governed by a comma. However the parser does not seem to be so strict. It has probably seen a few remaining cases in the training data (for whatever reason — most likely a coordination lacked the `Coord` s-tag) and it wills to draw such dependency whenever it thinks there is no better option. Our hard constraint simply ensures that any other option is considered better, no matter how many times the parser saw it.

If this constraint is turned off the accuracy will drop from 74.7 to 74.5 %.

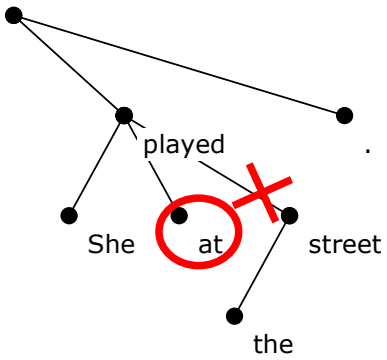


Figure 28: Example of prohibited skipping of a childless preposition.

8.12.6 No skipping of childless prepositions

In a sense this rule supports the fertility module. The fertility model takes care of the number of children of particular classes of words. For prepositions it almost guarantees that they will not get more than one child. It cannot however guarantee that each preposition gets its child. It may happen that another word steals the child from the preposition before the dependency preposition – child wins a round. To avoid that, it is prohibited to draw a left-to-right dependency over a childless preposition (i.e., a word right of a childless preposition depends on a word left of it). This embargo can be only withdrawn if the preposition gets a child or if there is no other dependency allowed.

If the constraint is turned off the accuracy will drop to 74.6 %.

8.12.7 No skipping of genitive nouns

A frequent construction in Czech is chaining noun phrases in genitive. The genitive case in this setting has roughly the same function as the English preposition "of". So the chain *cena opravy přední části střechy vozu prezidenta Spojených států amerických* would translate to "the price of the reparation of the front part of the roof of the car of the president of the United States of America" (common instances are shorter, though).

The Model Two learns that left-to-right dependencies D(N2, N2) are quite common, whether the words are adjacent or not (non-adjacent pairs occur whenever the dependent is modified by an adjective, like "the front part" in our example). Knowing that, the parser would easily connect two distant words labeled N2, jumping over another one. For instance, it could make "the president" depend on "the roof", bypassing "the car".

That is why we introduced a rule prohibiting such behavior. If an N2 is to be attached to another N2 to its left, there must not be a third N2 already lying in between and already attached to the leftmost member of the triple. In that case the rightmost N2 has to be attached to the N2 in the middle.

Recall that this hard constraint is an analogy to what has been "softly" learned from data for skipped verbs (see Section 8.6.7).

In final version of Model Two this rule is switched off. Otherwise the accuracy drops from 74.7 % to 74.6 %.

8.12.8 Relative clauses with wh-pronoun *který* "which"

PDT defines the following rules for relative clauses. The root of the clause is its main verb. The connecting wh-word usually depends on the verb, filling the gap after the noun that governs the clause. In a typical situation, there are three dependencies involved: 1) the verb hanging on a noun on its left; 2) the wh-pronoun *který* "which" lies between the noun and the verb, agrees with the noun in gender and number,[37] and hangs on the verb; 3) a comma lies left of the wh-pronoun, and hangs on the verb.

Example:

(23) *Uzávěrka přihlášek zájemců o účast v* ***misi, kterou povede*** *prezident Hospodářské komory ČR, je 10. září.* "10th September is the deadline for

[37] It does not however agree in case. The case of the wh-pronoun is driven by the verb.

applications of persons interested in participating in the **mission that will be led** by the president of the Business Chamber of the Czech Republic."[38]

It turns out that the Model Two is not particularly good in detecting such configurations. After all, we have mentioned that it considers one dependency at a time, not two or three. It learns that a verb can depend on a noun but it does not learn that it happens only if the verb dominates a relative clause or a speech.

A targeted hard rule, on the other hand, can be designed to look whether there is a form of *který* between the noun and the verb, and if so, it can check the agreement between the noun and the form of *který*. The rule can even check agreement in gender and number, although the statistics in Model Two are only able to check case agreements. If there are more than one agreeing nouns, the rule selects the nearest one. (In most cases this is the correct option. It will be wrong however if the found noun is the right member of a coordination.)

If this constraint is turned off the accuracy will drop to 74.6 %. There are 801 occurrences of forms of *který* in the d-test data; the rule applied 483 times, 389 times correctly.

8.13 Evaluation of the Model Two

8.13.1 Accuracy

The best-achieved accuracy of the Model Two on the PDT 1.0 d-test data is **74.7 %**. The following table summarizes contributions of all the respective features described in this chapter. The features occur in the same order they were described. The last row, "ALL SUCCESSFUL FEATURES", sums all features of the previous rows where the off-column displays lower accuracy than the on-column.

Feature	**Accuracy if turned off**	**Accuracy if turned on**
conditional probability	72.0	74.7
model-one-style tag reduction	70.1	71.0
model-two-style tag reduction	70.1	74.7
lexicalization λ=1	73.9	54.9
lexicalization λ=0.734375	73.9	74.7
selective lexicalization	72.2	
– the verb *být* "to be"	73.2	
– selected adverbs	74.2	
– pronouns	74.3	
– subordinative conjunctions	74.3	

[38] This sentence is the first example of a relative clause in the PDT 1.0 test data. Unfortunately there is a mistake in the PDT annotation of this sentence — PDT claims that the clause "that will be led…" depends on "deadline", not "mission".

Feature	Accuracy if turned off	Accuracy if turned on
– prepositions	74.5	
– verbs	74.6	
subcategorization	74.7	
jealousy (parameter of dependency weight)	74.7	64.3
no skipping of verbs	74.5	74.7
FFM (fertility)	74.7	74.5
TFM (fertility)		74.0
QFM (fertility)		
distance: adjacency	72.4	74.7
distance: adjacency and commas	72.9	
distance: weight /= *N*	74.6	
coordination	73.2	
short sentences	74.5	
n-tuple patterns	73.8	
hard constraints together	72.3	
– final punctuation	72.5	
– children of the root	74.6	
– root fertility	74.4	
– inter-comma segments	74.2	
– commas are leaves	74.5	
– no skipping of prepositions	74.6	
– no skipping of genitives	74.7	74.6
– relative clauses	74.6	74.7
ALL SUCCESSFUL FEATURES	**56.5**	**74.7**

The cross-evaluation of the Model Two parser on the PDT 1.0 e-test data yielded an accuracy of **74.9 %**. Such result sufficiently demonstrates that we have not stuck to the development data set too much. Besides this single run, the Model Two parser (as well as its author) had never seen the e-test data.

For the sake of comparison, we also ran the parser on the old (and unclean) PDT 0.5 data, both d-test and e-test.

Tested on / Trained on	PDT 0.5 d-test set	PDT 0.5 e-test set
PDT 0.5 training set	71.1	71.1
PDT 1.0 training set	73.0	72.9

As for Model One in Section 7.9.2, we evaluated the Model Two parser using various sources of morphological annotation. The results are shown in the following table. We only did not re-run the human-human test because it would cause the need of different test data (d-test data do not contain human-annotated morphology) and thus lessen comparability.

Some introduction to the particular morphological sources is given in the Sections 4.3 and 7.9.2. The architecture of the Model Two parser allows for further splitting the usage of the ambiguous dictionary morphology into two different ways. Either we can use the approach of Model One, i.e. to count 1/n of an occurrence of each tag X_i whenever a word occurred that could have been tagged by one of n tags $X_1 \ldots X_n$. Sum of relative frequencies of tags would be used during parsing. Or we can treat a sequence of possible m-tags as one string, one long tag.[39] The table refers to this approach as to *dictionary concatenated*.

Note: "NA" denotes experiments that were not run for the particular model.

Training	**Parsing**	**Model One**	**Model Two**
dictionary concatenated	dictionary concatenated	NA	71.1
dictionary	dictionary	51.4	67.8
tagger a	dictionary	NA	69.2
tagger b	dictionary	NA	69.2
human	dictionary	NA	69.2
dictionary	tagger a	53.7	72.9
tagger a	tagger a	**54.1**	**74.7**
tagger b	tagger a	NA	74.4
human	tagger a	53.4	73.9
dictionary	tagger b	NA	73.0
tagger a	tagger b	NA	74.5
tagger b	tagger b	NA	74.5
human	tagger b	NA	74.0
human	human	(51)	NA

8.13.2 Speed and memory

The parser has been rewritten in Perl and all statistics are now being stored in Perl hashes. Training the parser on PDT 1.0 takes about 13.5 minutes on an Intel Pentium 4, 1.8 GHz, 1.5 GB RAM, running Linux. Dividing the time by the 73088 non-empty sentences we get an average of 11 ms per sentence. The parsing of the 7319 test sentences takes about 30 minutes on the same machine, yielding an average of 246 ms per sentence (or roughly 4 sentences per second).

[39] For both approaches, duplicates are removed and list of tags ordered.

A rather large physical memory is essential because the parser easily consumes over 500 MB and swapping would unacceptably raise the time requirements. If enough physical memory is not available, better than swapping is to divide the training data into blocks, train statistics on each block separately, annotate each test sentence with all relevant statistics got from all blocks, and then run a modified version of the parser that sums up the statistics and builds the tree. Thanks to this trick we do not need to accommodate all the statistics at one time in memory while still being able to reflect them all in parsing one sentence. Of course we now need *n*-times plus constant more time where *n* is the number of blocks but it is still far better than using the system swapping mechanism that would result in an effectively endless process.

9 Automatic acquisition of subcategorization frames

In Section 8.6 we investigated several methods of using a machine-readable subcategorization dictionary to assist the Model Two Parser. We proved empirically that this parser had already been able to learn much of subcategorization on its own. However, it does not mean that subcategorization dictionaries are generally not useful for parsers. Carroll and Minnen (1998) and Carroll and Rooth (1998) give several reasons why subcategorization information is important for a natural language parser.

An electronic subcategorization dictionary can help in other fields of computational linguistics as well. It can serve machine translation to distinguish different meanings of verbs, forms the basis for tree families in tree adjoining grammars etc. Apart from that, subcategorization is interesting to lexicographers and linguists in general. It can be useful in discovering linguistic information about verbs (Siegel (1997)). For all those reasons it would be helpful to be able to enrich the PDT annotation and label dependents of verbs in PDT as either arguments or adjuncts.

Unfortunately only few languages have subcategorization dictionaries available, and if so, either these are not machine-readable, or they are not comprehensive enough to be used with a parser. Until recently, Czech was no exception.[40] Therefore a bunch of techniques for an automatic acquisition of such information from corpora have been developed.[41]

In Section 8.6 we used a list of subcategorization frames acquired by the approach proposed by Sarkar and Zeman (2000). In the present section we are going to briefly describe the algorithm that acquired the frames from PDT 0.5 training data.

Unlike other work in this area, this method does not assume that the set of subcategorization frames is fixed and known in advance. The frames are gathered from syntactically annotated data (PDT) where the subcategorization information is *not* given.[42]

9.1 Task description

In general, the problem of identifying subcategorization frames is to distinguish between arguments and adjuncts among the constituents modifying a verb. E.g., in "John saw Mary yesterday at the station" only "John" and "Mary" are required arguments while the other constituents are optional (adjuncts). There is some controversy as to the *correct*

[40] The first publicly available machine-readable Czech valence (subcategorization) dictionary, called VALLEX, contains only 1000 most frequent Czech verbs (annotated into depth and in high quality, though). There is also a list of frames found in PDT, called PDT-VALLEX, containing 5200 verbs. See Žabokrtský and Lopatková 2004 for VALLEX, and Hajič et al. 2003 for PDT-VALLEX.

[41] Methods for automatic acquisition of (not only) subcategorization from PDT have recently been published also in Bojar (2002) and Ondruška (2004).

[42] For those readers familiar with the PDT s-tags, it is important to note that the s-tag `Obj` does not always correspond to an argument in our sense. Similarly, the functional tag `Adv` does not always correspond to an adjunct. Approximately 50 verbs out of 2993 require an adverbial argument.

subcategorization of a given verb and linguists often disagree as to what is the right set of subcategorization frames for a given verb. A machine learning approach such as the one followed here sidesteps this issue altogether, since it is left to the algorithm to learn what is an appropriate subcategorization frame for a verb.

From now on, we will call the set of dependents (children) of a verb in PDT the **observed frame (OF)**. Note that which observed frame (or which part of it) is a true subcategorization frame (SF) is not marked in the training data.

Our task is twofold:

1. In an observed frame, find a subset that contains arguments only (Section 9.2).
2. Detect "corrupted" observed frames with missing arguments and filter them out (Section 9.3).

We are interested in surface frames, while the semantic distinctions that cannot be derived from the surface representation are not our priority. We do not aim to capture preferences such as that "to feed" prefers living objects, "to pour" needs a liquid etc. Consequentially, we also would not be able to realize common semantic class of some prepositions, such as locative (e.g. "in", "on", "behind"), neither their relation to some adverbs (e.g. "there"). Instead of getting to know that a verb (such as "to put") subcategorizes for a locative argument, we would get many low-density frames for all the respective instantiations of the argument.[43]

Most existing techniques for extracting SFs exploit the relatively fixed word-order of English to collect features for their learning algorithms using fixed patterns or rules. Such a techniques are not easily transported into a new language like Czech. Fully parsed training data can help here by supplying all dependents of a verb no matter where in the sentence these occur. The OFs obtained this way have to be *normalized* with respect to the word order, e.g. by using an alphabetic ordering.

9.2 Subsets of observed frames

Before we can apply any statistical methods to the training data, there is one aspect of using a treebank as input that has to be dealt with. A correct frame (verb + its arguments) is almost always accompanied by one or more adjuncts in a real sentence. Thus the *observed frame* will almost always contain noise. The approach offered by Brent (1991, 1993, 1994) and others counts all observed frames and then decides which of them do not associate strongly with a given verb. In our situation this approach will fail for most of the OFs because we rarely see the correct frames isolated in the training data. For example, from the occurrences of the transitive verb *absolvovat* "to go through

[43] In some SFs, a particular preposition is required by the verb, while in other cases it is a class of prepositions such as locative prepositions (e.g. "in", "on", "behind"…) that are required by the verb. In contrast, adjuncts can use a wider variety of prepositions. Prepositions specify the case of their noun phrase complements but a preposition can take complements with more than one case marking with a different meaning for each case (e.g. *na mostě* = "on the bridge"; *na most* = "onto the bridge"). In general, verbs select not only for particular prepositions but also indicate the case marking for their noun phrase complements (see also Section 8.6.1).

something" that occurred ten times in the corpus, less than 1/3 of occurrences consisted of the verb-object pair alone:

3× *absolvovat* N4
2× *absolvovat* N4 R2(od) R2(do)
1× *absolvovat* N4 R6(po)
1× *absolvovat* N4 R6(v)
1× *absolvovat* N4 R6(v) R6(na)
1× *absolvovat* N4 DB
1× *absolvovat* N4 DB DB

In other words, the correct SF constituted 30 % of the observed situations. Nevertheless, for *each* observed frame, one of its subsets was the correct frame we sought for. Therefore, we considered all possible subsets of all observed frames. We used a technique which steps through the subsets of each observed frame from larger to smaller ones and records their frequency in data. Large infrequent subsets are suspected to contain adjuncts, so we replace them by more frequent smaller subsets. Small infrequent subsets may have elided some arguments and are rejected.

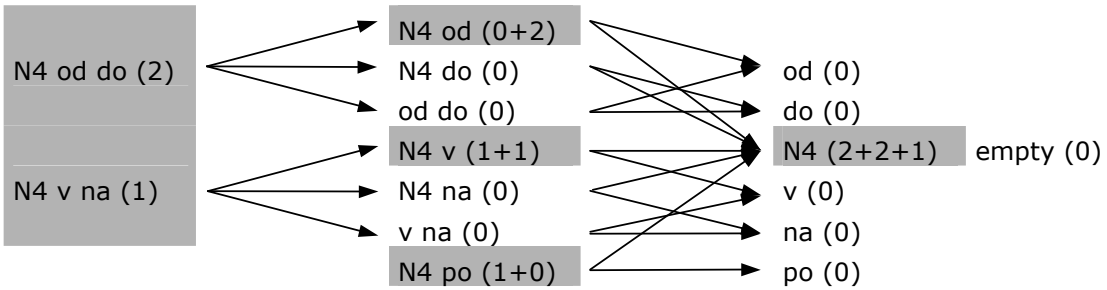
Initially, we consider only the observed frames (OFs) from the treebank. There is a chance that some are subsets of some others but now we count only the cases when the OFs were seen themselves. Let's assume the test statistic rejected the frame. Then it is not a real SF but there probably is a subset of it that is a real SF. So we select one of the subsets whose length is one member less: this is the *successor* of the rejected frame and inherits its frequency. (If the successor itself occurred in the training data as OF, the inherited frequency would be added to its own frequency.) Of course one frame may be a successor of several longer frames and if so, it inherits frequencies from all of them. This is how frequencies accumulate and frames become more likely to survive.

An important point is the selection of the successor (or the dependent to be removed from a rejected frame). We have to select only one of the n possible successors of a frame of length n, otherwise we would break the total frequency of the verb. Since we aim to choose the most frequent subpart, we wait until the counts of the frames on lower level are definite, i.e. until all the frames of the current length are processed. At that point we have m rejected frames of length n, each of which can be shortened in n ways. This yields $m \times n$ possible modifications to consider before selection of the successor. We implemented two methods for choosing a single successor frame:

- Choose the one that results in the strongest preference for some frame (that is, the successor frame results in the lowest entropy across the corpus). This measure is sensitive to the frequency of this frame in the rest of the corpus.
- Random selection of the successor frame from the alternatives.

Random selection resulted in better precision (88 % instead of 86 %). It is not clear why a method that is sensitive to the frequency of each proposed successor frame does not perform better than random selection.

After processing a frame we do the same with next one of the same length. Once there is no such frame we descend to the level of frames one dependent shorter and so on, until all frames have been processed.



Computing the subsets of observed frames for a verb. The counts for each frame are given within parentheses. In this example, the frames N4 R2(od) R2(do), N4 R6(v) R6(na), N4 R6(v), and N4 R6(po) have been observed with the given verb in the corpus.

9.3 How to reject OFs

The last important point to explain is how we measure whether a frame is to be accepted or rejected. For each verb and a particular set of its dependents we need to associate a score to the hypothesis that the dependents are arguments of the verb. In other words, we need to assign a value to the hypothesis that the OF under consideration is the verb's SF. Intuitively, we either want to test for independence of the observed frame and verb distributions in the data, or we want to test how likely is a frame to be observed with a particular verb without being a valid SF.

Sarkar and Zeman (2000) tested three statistical approaches to testing the hypothesis: a likelihood ratio test, t-scores, and standard binomial hypotheses testing. We are going to present the last one as they proved it to be most successful. The subcategorization dictionary used in Section 8.6 has also been acquired using this approach.

Assuming that the data is binomially distributed, we can look for frames that co-occur with a verb more often than chance. This is the method used by several other papers on SF extraction starting with (Brent (1991), Brent (1993), Brent (1994)).

Let us consider probability $p_{\neg f}$, which is the probability that a given verb is observed with a frame but this frame is not a valid SF for this verb. $p_{\neg f}$ is the error probability on identifying a SF for a verb. Let us consider a verb *v* which does *not* have as one of its valid SFs the frame *f*. How likely is it that *v* will be seen *m* or more times in the training data with frame *f*? If *v* has been seen a total of *n* times in the data, then $H^*(p_{\neg f}; m, n)$ gives us this likelihood.

$$H^*(p_{\neg f}; m, n) = \sum_{i=m}^{n} p_{\neg f}^{i} (1 - p_{\neg f})^{n-i} \binom{n}{i}$$

If $H^*(p; m, n)$ is less than or equal to some small threshold value then it is extremely unlikely that the hypothesis is true, and hence the frame *f* must be a SF of the

verb *v*. Setting the threshold value to 0.05 gives us a 95 % or better confidence value that the verb *v* has been observed often enough with a frame *f* for it to be a valid SF.

9.4 Evaluation

The method has been evaluated using the PDT 0.5 training data. In this training set, there were 33,641 verb tokens with 2,993 verb types. There were a total of 28,765 observed frames, which reduced to 13,665 observed frames after preprocessing. There were 914 verb types seen 5 or more times.

Since there was no electronic subcategorization dictionary for Czech, we evaluated the filtering technique on a set of 500 test sentences, which were unseen and separate from the training data. These test sentences were used as a gold standard by distinguishing the arguments and adjuncts manually. We then compared the accuracy of our output set of items marked as either arguments or adjuncts against this gold standard.

The results can be compared to two baseline methods. Baseline method 1: consider each dependent of a verb an adjunct. Baseline method 2: use just the longest known observed frame matching the test pattern. If no matching OF is known, find the longest partial match in the OFs seen in the training data. We exploit the functional and morphological tags while matching. No statistical filtering is applied in either baseline method.

The following table compares the baseline methods and the statistical filtering described above. Some of the values are not integers since for some difficult cases in the test data the value for each argument/adjunct decision was set to a value between [0,1]. *Recall* is computed as the number of known verb complements divided by the total number of complements. *Precision* is computed as the number of correct suggestions divided by the number of known verb complements. $F_{\beta=1} = \frac{2 \times p \times r}{p + r}$. *% unknown* represents the percent of test data not considered by a particular method.

	Baseline 1	**Baseline 2**	**Hypothesis testing**
Precision	55%	78%	88%
Recall	55%	73%	74%
$F_{\beta=1}$	55%	75%	80%
% unknown	0%	6%	16%
Total verb nodes	1027	1027	1027
Total complements	2144	2144	2144
Nodes with known verbs	1027	981	907
Complements of known verbs	2144	2010	1812
Correct suggestions	1187.5	1573.5	1596.5
True arguments	956.5	910.5	834.5
Suggested arguments	0	1122	674
Incorrect arg. suggestions	0	324	27.5
Incorrect adj. suggestions	956.5	112.5	188

Our method discovered 137 subcategorization frames from the data. The known upper bound of frames that the algorithm could have found (the total number of the observed frame types) was 450.

The technique described here may sometimes find a subset of a correct SF, discarding one or more of its members. Such frame can still help parsers because they can at least look for the dependents that have survived.

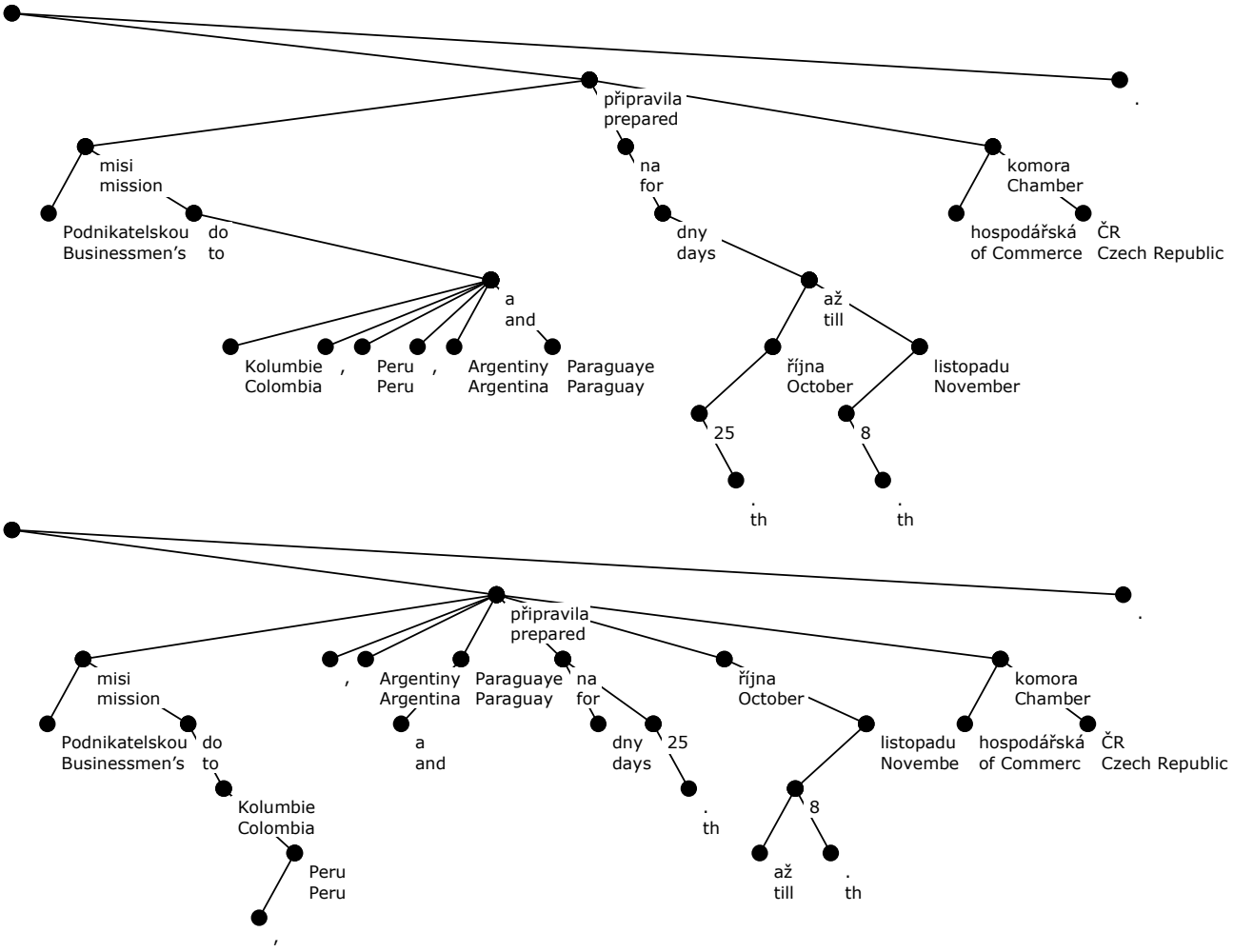


Figure 29: The upper tree is the correct parse of the sentence (24). The lower tree is the output of the parser for the same sentence when there are tagging errors.

10 Morphology disambiguation: poor output of taggers and workarounds

Accuracy of a parser that relies on morphological tags essentially depends on the success rate of the tagger. Any tagging error may violate agreement in gender, number or case, often the determining factors for syntactic relations. The case tagging errors are the most crucial.

Hajič and Hladká (1998) published the error rate of the maximum entropy tagger (tagger "a"): 6.2 %, i.e. the accuracy is 93.8 %. They also measured error rate of separate morphological attributes of each word. For us is important the accuracy of predicting the attributes used in our reduced tag set: case (95.2 %) and subpart of speech (99.5 %). We counted the same on our training data.[44] We got 92.6 %[45] overall

[44] We could not include our test data in the experiment because it does not contain manually annotated morphology.

for tagger "a", and 92.7 % for tagger "b". Both figures are better when our tag reducing scheme (see Sections 7.2 and 8.4, and Hajič et al. (1998)) is applied: 94.5 % "a", 94.4 % "b". On the other hand, assigning the correct case is one of the more difficult tasks of tagging. We tested only the words whose correct case was known (i.e. 1-7, not X nor –): 91.7 % "a", 91.5 % "b". From the point of view of the parser it is also interesting in how big part of *all* words we can expect a case tagging error. The correct cases divided by all words give 95.3 % "a" and 95.2 "b".

Now let us look at an example:

(24) Podnikatelskou misi do Kolumbie, Peru, Argentiny a Paraguaye připravila na dny 25. října až 8. listopadu hospodářská komora ČR. "The businessmen's mission to Colombia, Peru, Argentina, and Paraguay, was prepared for the days 25th October till 8th November by the Chamber of Commerce of the Czech Republic."

The four names of South American countries are coordinated and agree in genitive case required by the preposition *do* "to". The tagger was confused and assigned genitive, unknown, genitive, and nominative, respectively. No wonder that the parser was confused even more and constructed a useless tree structure. Figure 29 shows the correct parse of the sentence (24) (upper tree), and the output of the parser for that sentence (lower tree).

As shown in Section 8.13.1, simple switching to non-disambiguated morphological annotation would not help. There is a possible workaround in incorporating a tagging or partial-tagging procedure into the parser. Whenever the parser selects a dependency the morphological situation of the words involved gets a bit clearer. For instance, if the dependency observes agreement in case, and the words allow tag combinations N1|N4|N5 and N1|N2|N4, the pairs N1–N1 and N4–N4 will contribute with much higher probability amount than the other combinations. Thus the probability of the words having one of the tags N1, N4 (as opposed to N2 and N5) will get higher. Such information could than be used in finding the other dependencies. This might overcome the advantage of chart parsers that can naturally incorporate tagging into parsing.

[45] The tagging accuracy is measured on all words, not only the ambiguous ones but also words where the tagger had nothing to solve. Such accuracy is important from the point of view of a parser, which needs to know how good is its input. If we tested only the ambiguous words, we could say that the tagger was successful at 88.0 %. However, we still would not be able to distinguish between the words where the tagger had to choose one of two possibilities and the words where there were two dozens of choices.

11 Regular expressions and statistics

Zeman (2001a, 2001b) reports on experiments with combining the Model One statistical parser with a pre- or postprocessor based on regular expressions (RE). The expressions were handcrafted and they described rules for partial parsing of some Czech syntactic constructions. The rules were capturing chunks that could theoretically be unlimitedly complex but in real world they rarely were.[46] The application of the rules was not completely error-free, nevertheless the rules were selected so that their accuracy on PDT training data would not drop under 90%.

Several rules were applied to one sentence consecutively, and one rule was applied recursively as long as there was material for it. That effectively made the power of the whole vehicle context-free.

There were two or three ways how to combine the RE-based partial parser with the statistical full parser. The first one assumed that the preprocessor would replace each recognized phrase by a representative word (typically by its head but coordinations — whose head was a conjunction — should have been represented by one of their members, converted to plural). The parser then would see the preprocessed phrases neither during the training phase, nor during parsing. Such phrases became atomic items for the parser.

There is a possible drawback of this method. If the preprocessor fails to find all members of a phrase, the error can be corrected by the parser only if the forgotten member depends on the head of the phrase. If it ought to be nested more deeply in the phrase, its real governor is now invisible because the phrase is atomic. This leads to the second approach where the phrase would even for the parser be a structure rather than a monolith. The parser would get some dependencies for free but would be allowed to add new dependencies at any place in the structure. The third approach is a special case of the second one. It applies the finite state tool as a postprocessor to the statistical parser output, overriding all parser decisions concerning the recognized chunks.

We refer to the three methods as to **transparent preprocessing**, **non-transparent preprocessing**, and **postprocessing** respectively.

The recall of the rule-based tool was almost 20.0 %, which meant that it would be able to (correctly) help in one fifth of cases. The precision of the tool was 94.0 %. If the preprocessor used morphological markup by the tagger "a" instead of hand annotation, the precision would drop to 92.3 %, recall to 17.3 %. If ambiguous output of morphological analysis were used, precision would be 89.5 %, recall 20.1 %.

Finally we combined the finite state tool with the parser in all three ways described above. The resulting systems were tested on PDT 0.5 d-test data. All experiments used machine-disambiguated morphology.

[46] We used regular expressions to model the local syntax of coordinations, simple noun phrases and other simple small chunks of sentences. The method is closely related to *local grammars* that have been applied to a similar task for a similar language (Serbo-Croatian) by Nenadić and Vitas 1998, and Nenadić 2000.

Besides speeding up the parsing process (most significantly with transparent preprocessing), Zeman (2001b) reported increased accuracy. However, those days' numbers were far from Model Two final performance: the accuracy rose from 53.7 to 57.2 with nontransparent preprocessing. It was not sure whether at least some improvement could be achieved for a parser whose accuracy would be 70 % or more. For the Model Two parser, the answer is no, for a simple reason: the phenomena captured by the RE rules are simple enough so that the parser can learn them now. So if parsing time is not an issue, there is no reason more for using the preprocessor.

12 Parsing non-projective constructions

In Section 7.1 we defined projectivity[47] and required that the parser render all trees projective. At the same time we demonstrated that there are correct non-projective sentences but they are so rare that enforcing projectivity would not damage the results too much.

Theoretically, the Model Two parser already *can* produce limited amount of non-projective constructions. It comes with the special treatment of some patterns and short sentences (see Sections 8.10, 8.11) where a non-projective subtree can be learned *en bloc*. Practical effect is negligible: only two non-projectivities in two sentences appeared in the output so far.

The state-of-the-art Czech parsers of Eugene Charniak and Michael Collins respectively[48] so far treat Czech as being completely projective. They are not able to produce non-projective output (it follows from the fact that they first generate context-free-grammar-based parse trees and then convert them to dependencies). During training they simply shift any non-projectively attached node to a nearest projective position so that it is possible to convert the dependency structure into parse tree.

There are other parsers (Žabokrtský, Holan) that *are* able to generate non-projective constructions.[49] Unfortunately, description of these parsers has not been published. See Chapters 14 and 15 for a little more about them.

Holan et al. (1998), and Kuboň et al. (2001) develop grammar formalisms that allow for handling of non-projectivities. We do not know about any parsing system based on their formalism that would have been evaluated on PDT d-test data.

Non-projective constructions are not much frequent in terms of word counts but are relatively frequent in terms of sentence counts. Out of the 73,088 non-empty sentences in PDT 1.0 training data, 16,920 (23.2 %) contain one or more non-projective dependency. Out of the 1,255,590 words in the same data set, 23,691 (1.9 %) are attached non-projectively. Both percentages are quite close to the figures reported in Chapter 2 of Hajič et al. (1998) for PDT 0.5.

We have categorized the main groups of non-projectivities according to their cause. The statistics taken from PDT indicate that majority of non-projectivities found in real data is of technical nature. That gives us a chance to teach the parser to recognize non-projective constructions.

---

[47] Several mutually equivalent definitions of projectivity (Hudson's (1984) adjacency) have been formulated; see esp. Marcus (1965).

[48] See Hajič et al. (1998) and Collins et al. (1999) for Collins' parser. For Charniak's English parser see Charniak (2000); Charniak adapted it for Czech in 2002 and during his visit to Prague in January 2003; however, to our knowledge the results have not been published.

[49] We found 2377 non-projective dependencies in 1485 sentences of the output of Žabokrtský's parser. For Holan's parsers it is 261 words / 147 sentences (l2r parser), 224 / 153 (r2l parser), and 1518 / 906 (pshrt parser). The hand annotation of PDT 1.0 d-test data contains 2306 non-projectivities in 1650 sentences.

12.1 Classification of non-projective constructions in PDT

Non-projective constructions form a relatively little explored part of (Czech) language. A classification has been proposed by Uhlířová (1967). However, she did not have a syntactically annotated corpus at her disposal, which yields two consequences. On one hand, she does not provide any idea how frequent this or that type of non-projectivity is. On the other hand, she does not have to bother with technical cases, bound to the treebank annotation guidelines. Some observations about non-projectivities from a parser's point of view are described in Holan (2003), though they bring just statistics about different POS-tag configurations.

We[50] have recently developed a preliminary classification based on PDT data. According to that classification, three main groups are to be distinguished, in which the non-projective constructions concern:

(A) combinations of lexical units with **function words** (especially auxiliaries), which correspond to no non-projectivities in the tectogrammatical level, since in the latter such a combination is represented by a single node;

(B) syntagms divided, under the surface word order, into a contextually non-bound part and a (generally contrastive) contextually bound part, the latter being transferred to the left;

(C) phrasemes consisting of more than one surface word, which eventually are to be treated as not containing a dependency relation in the tectogrammatical level (either each of them is to be specified by a single node of the tectogrammatical level, or by a specific relation, different from syntactic dependency).

Let us now illustrate the three groups by examples and statistics taken from the PDT 1.0 training data. The technical non-projectivities (class A) are of the most interest at the time being, as we expect to be able to incorporate these into the parser with least effort. We will see that about a half of all PDT non-projectivities falls into the class A.

Before we proceed, it will be useful to define a **gap**. First recall the condition of projectivity. In a projective rooted tree the following implication holds (*S* denotes the relation of subordination (*aSb* indicating that *a* is subordinated to *b*), an irreflexive transitive closure of dependency; the set of nodes is ordered by the relation *W*.):

$$\forall x, y, z, v \left((xSz \wedge ySz \wedge xWv \wedge vWy) \Rightarrow vSz \right)$$

We say that a node *z* for which *vSz* in the implication does not hold is in a **gap**. For more on gaps and for a discussion of the possibilities of several gaps co-occurring in a sentence, see Kuboň et al. (2001), and Holan et al. (1998).

12.1.1 The conjunction *-li* (A1a)

The Czech conjunction *–li* "if" occurs in a specific position: after a verb that starts the clause; if the verb is followed by a dependent, then *–li* is in a gap and a non-projectivity follows, or several of them at once.

[50] The presented classification is a result of my research in collaboration with Kateřina Veselá and Petr Sgall. A comprehensive version is going to appear as a technical report in the near future (see also Veselá et al. 2004, Hajičová et al. 2004).

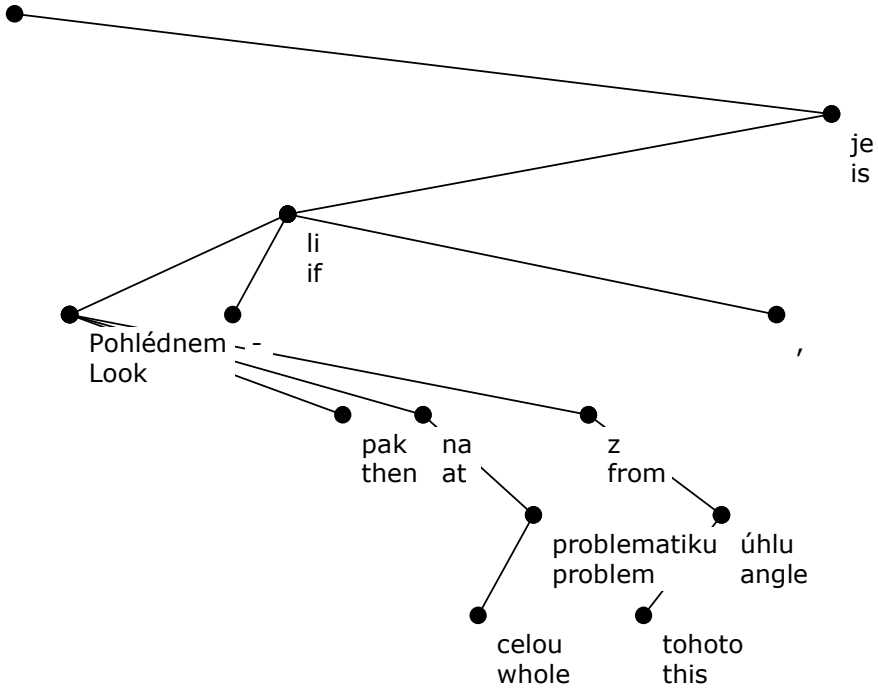


*Figure 30: Non-projective –*li*:*
Pohlédnem-li pak na celou problematiku z tohoto úhlu, …
If we view the whole problem from this angle…

(25) Pohlédnem-li pak na celou problematiku z tohoto úhlu, … "If we view the whole problem from this angle…"

This part of the A1 class forms about 4.6 % of non-projectivities found in PDT.

12.1.2 Auxiliary verb forms (A1b)

Here belong examples such as

(26) Bude to muset udělat hned. "He will have to do it at once."

Since on tectogrammatical level the function words (as the auxiliaries *bude* for future tense, and *muset* for the modality) are rendered by indices of their lexical verbs, rather than by extra nodes.

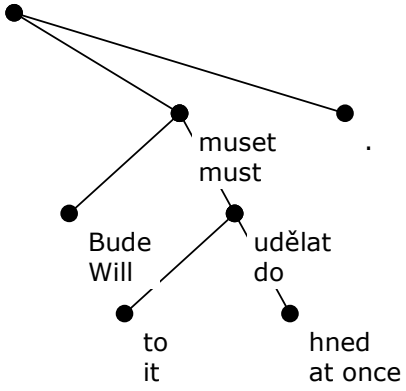


Figure 36

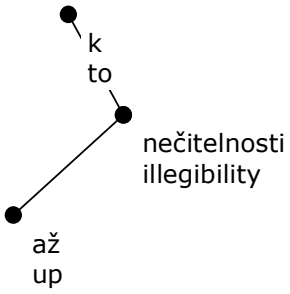


Figure 31: Focus-sensitive particle

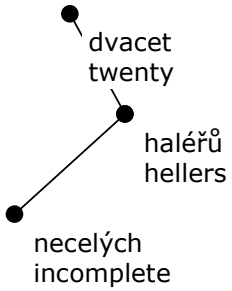


Figure 32: Numeral in the gap

The A1 class together forms about 27 % of non-projectivities found in PDT.

12.1.3 A prepositional group with a focus sensitive particle (A2)

(27) až k nečitelnosti "up to illegibility"

The node dependent on the noun is a focus sensitive particle (focalizer, rhematizer), which has just the noun in its domain, although it precedes the preposition (the gap).

The following table brings examples of the most frequent focalizers in the PDT 1.0 training data. A total of 6124 occurrences of 182 focus sensitive particles have been encountered. For the sake of this statistics, the system marked as focalizer every node crossing a preposition and hanging on the word behind it. Some of such "focalizers" are more likely to be attached elsewhere in the described configurations; 48 of them (in 5546 occurrences) have prevailingly (more than 50%, more than 1 occurrence) attached to the second node to the right, crossing the preposition.

Focalizer	**Translation**	**Examples**	**Counterexamples**	**Percentage**
i	as well	1373	900	60.4
až	up to	719	316	69.5
jen	only	441	197	69.1
pouze	just	278	128	68.5
již	yet	252	154	62.1
už	already	227	181	55.6
především	first of all	222	140	61.3
ještě	still	209	139	60.1
zejména	in particular	200	128	61.0
ani	not a single	166	151	52.4
…				
jenom	only	28	7	80.0
skoro	almost	9	1	90.0
např.	e.g.	52	0	100.0

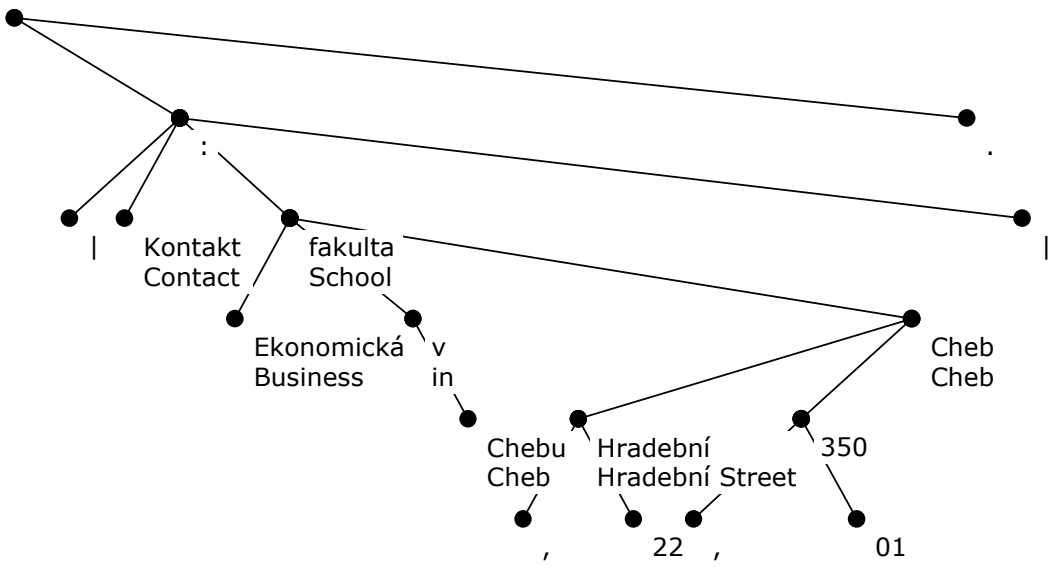


Figure 33: Bracketed sentence:
| Kontakt: Ekonomická fakulta v Chebu, Hradební 22, 350 01 Cheb. |
| Contact: Business School in Cheb, Hradební 22, 35001 Cheb. |

The A2 class forms about 26 % of non-projectivities found in PDT.

12.1.4 A numerative handled as a noun, rather than an adjective, and expounded then by a divided noun group

(28) necelých dvacet haléřů "less than twenty hellers"

Due to the agreement in case between the noun *haléřů* and the adjective *necelých,* the latter is analyzed as depending on the former, which itself depends on the numeral. On the tectogrammatical level, the numeral is understood as a (syntactic) adjective, depending on *haléřů,* so that the condition of projectivity is met.

The A3 class forms about 0.6 % of non-projectivities found in PDT.

12.1.5 Bracketed sentences (A4)

If the whole sentence is in brackets or similar pair symbols (such as quotes or even vertical bars), the final stop (question mark, exclamation mark) causes a technical non-projectivity, as it, according to the annotation guidelines, hangs on the root, while the brackets hang on the main verb.

(29) | Kontakt: Ekonomická fakulta v Chebu, Hradební 22, 350 01 Cheb. |
"| Contact: Business School in Cheb, Hradební 22, 35001 Cheb. |"

Odd it may be, such construction is nevertheless not so rare: we counted 959 occurrences (4 %).

12.1.6 Non-projectively attached punctuation and filler words (A5)

All other non-projective dependencies whose dependent node's s-tag is AuxG, AuxX, AuxY, or AuxZ belong here. The group is rather heterogeneous and does not provide a simple

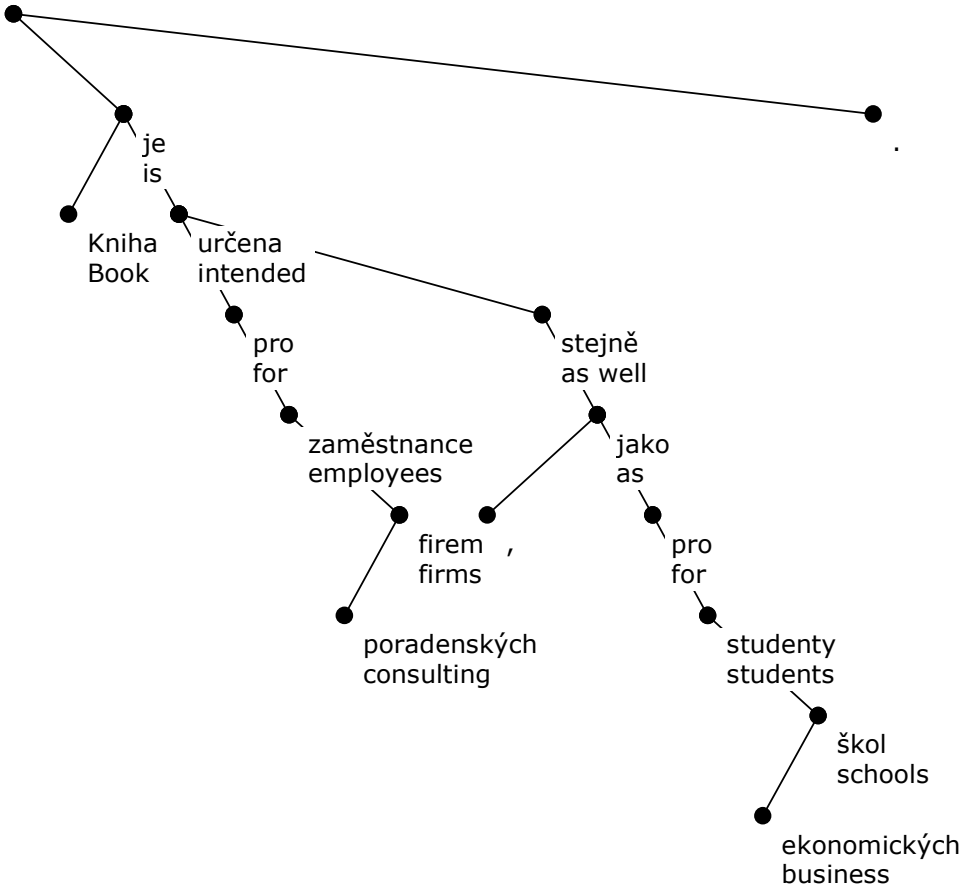


*Figure 34: Non-projectively attached punctuation: **,** stejně jako* "as well as" *in sentence (30).*

clue for the parser to parse it. For example we chose a non-projectively attached comma (AuxX):

(30) Kniha je určena pro zaměstnance poradenských firem, stejně jako pro studenty ekonomických škol. "The book is intended to assist employees of consulting firms as well as students of business schools." (c122:42)

12.1.7 Coordination with an adjunct depending on the group as a whole (B1)

(31) …, kdy si zahraniční banky najdou cestu přímo do regionů, případně kdy si začnou zvát zajímavé klienty do své pražské pobočky. "… when foreign banks find their way to the regions, or when they start to invite attractive clients to their subsidiary in Prague."

The conjunction *případně* "or" coordinates the clauses, *banky* "banks" is the common subject of the whole coordination and thus hang on *případně*, the words *kdy* "when" and *si* "themselves" are therefore separated from their parent node *najdou* "find".

The B1 class forms about 5 % of non-projectivities found in PDT.

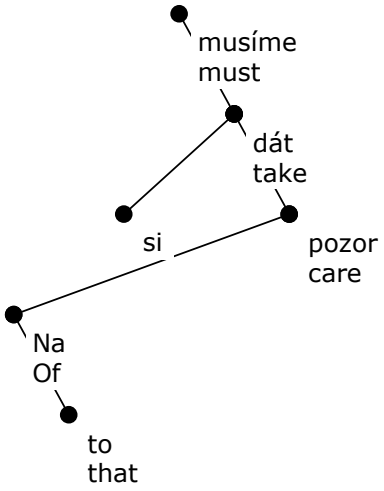


Figure 35: Dislocated dependent of a phraseme

12.1.8 Phrasemes with a dislocated dependent (B2)

(32) Na to si musíme dát pozor. "We have to take care of that."

The phraseme *dát pozor na* "to take care of" can be understood to be interrupted here, since the *of* group is a part of the topic and its head noun is in the focus.

The B2 class forms about 0.1 % of non-projectivities found in PDT.

12.1.9 Divided nominal groups (B3)

(33) Společnou máme především tuto zodpovědnost. "First of all it is this responsibility what we have in common."

The adjective *společnou* is pre-posed as a contrastive adjunct of the contextually non-bound subject.

The B3 class forms about 10 % of non-projectivities found in PDT.

12.1.10 Numerals with a dislocated dependent (B4)

(34) Běžně je jich k dispozici deset. "Commonly, ten of them are at disposal."

The group *jich deset* "ten of them" is divided by the prepositional group *k dispozici*, which depends on the verb.

The B4 class forms about 1.1 % of non-projectivities found in PDT.

12.1.11 A comparative group divided from its 'than' dependent by its headword (B5)

Recall the example sentence (13) in Section 7.1:

(35) … protože doba přenosu více závisí na stavu telefonní linky než na rychlosti přístroje. "… because the transmission time depends more on state of the phone line than on the speed of the device."

We do not repeat the corresponding Figure 12 here, please see the Section 7.1 for it.

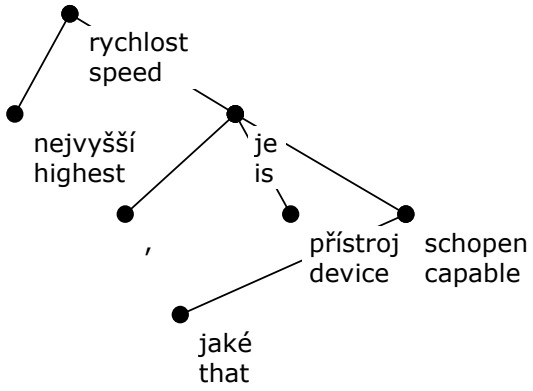


Figure 36: Dislocated wh-element

A "comparative group" need not necessarily use morphological means of the comparative degree. The positive or superlative degrees can be present as well, as in the following examples:

(36) podobný pes jako sousedův

"a similar dog as the neighbor's one"

(37) nejrychlejší běžec na světě

"the fastest runner in the world"

The B5 class forms about 2.6 % of non-projectivities found in PDT.

12.1.12 Relatives or interrogatives (wh-elements) dislocated to the left (B6)

(38) nejvyšší rychlost, jaké je přístroj schopen

"the highest speed the device can achieve"

The wh-pronoun depends on a nominal part of the predicate, and the headword (possibly with other dependents) is in the gap. A similar behavior is proper to wh-words in interrogative dependent clauses.

The B6 class (including its intersection with B7) forms about 1.9 % of non-projectivities found in PDT.

12.1.13 Dislocated dependents of infinitives (B7)

Recall the example sentence (12) in Section 7.1:

(39) Soubor se nepodařilo otevřít. "The file could not be opened."

We do not repeat the corresponding Figure 11 here, please see the Section 7.1 for it.

Quasi-modal predicates are placed here in the "second" (weak) position, after a possible (sequence of) clitic(s), perhaps in analogy to real modal verbs (cf. the A1 class), which, in the prototypical case, occupy the weak position.

Besides quasi-modal predicates, some other are involved in this class, such as phase- or quasi-phase predicates. Some examples of frequent gap words governing the infinitives are *lze* "it is possible", *začít* "begin", *hodlat* "intend", *podařit* "manage", *nechat*

"let", *snažit* "try hard", *schopný* "able", *přestat* "cease", *pokusit* "attempt", *potřebovat* "need", *odmítat* "refuse", *ochotný* "willing", *povinný* "required" etc.

The B7 class (excluding its intersection with B6) forms about 8 % of non-projectivities found in PDT.

12.1.14 Particles referring to preceding co-text, although occupying the 2nd position (B8)

Czech particles such as *však* "however", *proto* "therefore" are understood on the analytical level as heads (with the verb depending on them); they prefer to occur in a gap. They generally refer to the preceding co-text. It is like if there was a coordination of clauses, split into two sentences, with the conjunction (*však*) residing staying with the second part.

(40) Na tom však vinu nemám. "However I'm not guilty for that."

The B8 class forms about 16 % of non-projectivities found in PDT. The most frequent gap words are *však* "however", *ale* "but", *proto* "therefore", *ovšem* "admittedly".

Even our enumeration is not exhaustive; we have focused mainly on the A-class technical subclasses, and have omitted some low-density classes, such as separated members of an asymmetrical apposition, nominal vs. verbal attributes (cf. Uhlířová (1967)), deletions etc.

We have shown that at least a half of the non-projective constructions in real data is rather technical and thus should be easily solvable by parsers.

12.2 Treating non-projectivities by the parser

Now as we have rough descriptions of the situations, in which non-projectivities typically emerge, we can modify the parser to recognize at least some such situations and to lift the ban of non-projectivity with some care.

A purely statistical way would be to look into the training data and learn patterns that can be solved non-projectively. Unfortunately, the thing gets complicated due to sparse data: the patterns would be described partially by m-tags, sometimes by lexical values (lemmas or word forms), and if we want the process to be guided solely by the data, we cannot direct the parser where to use word forms and where the lemmas or m-tags.

As a solution, we split treating non-projectivities in two parts. Compact constructions, well describable by m-tag patterns, are analyzed as blocks, together with some projective structures — see (8.11). The necessary condition is that the pattern has been observed at least five times and that there is a dependency structure that applies to the pattern in at least 90 % of the examples found in the training data.

Non-projectivities that cannot be captured by the above statistical procedures will be subject to fine-tailored, hand-made rules based on the classification in this section.

12.2.1 Conjunctions in the second position

The *–li* conjunction in 0 (including the dash) can always be non-projectively crossed by any dependency on the left-neighboring verb. Either we can modify the function

responsible for allowing dependencies for addition, or we can leave the function as is, reorder the words so that *–li* precedes the verb, parse the sentence and reorder it back again.

Similar approach could help with *však* and other conjunctions in second positions (see 12.1.14). Here it might be more useful to remove the conjunction from the sentence temporarily, instead of reordering. After parsing we would insert the conjunction on the edge going from the root to the main verb.

We have implemented the light version, i.e. the projectivity-watch function just allows crossing *–li* and *však*, and the rest is left for the statistical core to decide.

12.2.2 Constructions with infinitives

Infinitive verbs get involved in many non-projectivities (cf. 12.1.2 and 12.1.13). A good recipe might be to review any parse where an infinitive is present. If an argument is more strongly lexically bound to the infinitive than to the finite verb above it, but the projectivity enforcement bans such dependency, allow it! Attention, subjects should be attached to the governing finite verb.

We have implemented just a simple version, which relies more on the statistical core. If the parser has already attached an infinitive to its left neighbor, all nodes that are allowed to attach to the neighbor will now be also allowed to attach to the infinitive.

12.2.3 Prepositions and focalizers

The A2-class non-projectivities (see 12.1.3) are quite compact (usually no more than three nodes involved) and their patterns could easily be learned. However, the general pattern-matching tool described in 8.11 cannot apply here as long as it checks m-tags only. A list of focus-sensitive particles (their lemmas) learned from the training data will fill the gap.

The parser is now looking for a focalizer from the list, followed by a preposition. Given the statistics presented in 12.1.3, enforcing the attachment of the focalizer to the word after the preposition would not help much if at all. So we only allow the non-projective dependency to compete. The main statistical model selects whether the dependency wins over the other possibilities of attaching the focalizer-resembling node.

12.2.4 Results

Experiments revealed that the partial processing of non-projectivities we allowed has a very limited, though positive influence on the accuracy. The output from the parser now contains 388 non-projective dependencies in 322 sentences. If processing non-projectivities was turned off, the accuracy would drop from 74.7 to 74.5 %.

13 Extended evaluation methods

This chapter is a follow-up to Sections 7.9 and 8.13 that evaluate the accuracy of Models One and Two, respectively. Evaluation in those sections mainly involves comparing the influence of various parser features on the accuracy. In contrast, in this chapter we present only tests of the *best parser* (all successful features turned on) but viewed from many new perspectives.

13.1 Accuracy on sentences of different lengths

The accuracy on short vs. long sentences has been first discussed in the Section 8.10. In the following table we summarize the accuracies on sentences of different lengths for the final version of Model Two.

Length	# Sentences	# Words	Accuracy	
1 – 10	2363	14334	84.8	75.7
11 – 20	2727	41819	77.0	
21 – 40	2125	57970	72.4	
41+	255	12360	66.4	

13.2 Sentence accuracy

So far the term *accuracy* referred to *dependency accuracy*, i.e. the number of correctly attached words divided by the total number of words in the test data. **Sentence accuracy**, on the other hand, is defined as the number of correctly parsed sentences divided by the total number of non-empty sentences in the test data. A **correctly parsed sentence** has all words attached correctly, i.e. its dependency accuracy is 100 %.

Some applications may prefer to get only some parses with guaranteed quality over getting all parses with average quality. That is why sentence accuracy may be a useful measure.

It is extremely difficult to correctly parse a long sentence. Of course, it is easier for short sentences, and sentences of length 1 are correct automatically. The total number of sentences in the PDT 1.0 d-test data is 7319. The total number of correct parses delivered by the Model Two parser is 1539. Thus the sentence accuracy is **21.0 %**. Distinctions according to the sentence length are demonstrated in the table below; the longest correctly parsed sentence contains 32 words.

Length	Sentences	Correct parses	Sentence accuracy
1	55	55	100.0
2	199	198	99.5
3	151	122	80.8
4	209	148	70.8
5	274	196	71.5

Length	Sentences	Correct parses	Sentence accuracy
6	220	118	53.6
7	276	128	46.4
8	248	93	37.5
9	267	98	36.7
10	313	85	27.2
…			
32	70	1	1.4
Total	**7319**	**1539**	**21.0**

We can modify the word-based accuracy to reflect the sentence accuracy as well. We would consider a word correct only if it appeared in a perfectly parsed sentence. As usual, the sum of correct words would be divided by the total number of words. In other words, this is a sentence accuracy weighed by the sentence length.

The 1539 correct parses together contain 10716 words. As the total number of d-test words is 126030, the weighed sentence accuracy is **32.1 %**.

13.3 Parser skillfulness

One may tend to underrate "decisions" in cases where in fact there was little or nothing to decide. For instance, if a tagger has to tag a word that is covered by dictionary and only allows for one tag, one may claim that the tagger should get no score points to its accuracy figure for that word because there was nothing to disambiguate. Similarly, one may claim that "sentences" consisting of one single word should not be counted in parser accuracy.

There are two problems with such a point of view. First: for an application using the output being evaluated, it does not matter how difficult the job of the tagger (parser, or another tool) was. The only thing that matters is the quality of the *whole* data the application gets — including the easy points. And second: there is a difficulty spectrum rather than a two-state classification (*easy* vs. *difficult*). If we penalize (or leave out) cases where a tool has nothing to solve, how will we distinguish the cases where it has to choose one of two values (tags, dependencies…) from cases where one of 20 values is to be selected?

We attempt to address the issue in this section. We develop a measure that is based in parsing accuracy but applies weights according to the difficulty of each parsing decision. To distinguish the measure from *parsing accuracy*, we call it **parser skillfulness**. We do not develop it very deeply though; this is just an illustration of one possible way.

Let's assume we have a measure of difficulty of attaching word, denoted O_w and being a function of a word w. Then the parser skillfulness S on test data DTEST would be defined as

$$S = \frac{C_{OK}}{C}, \text{ where } C_{OK} = \sum_{\substack{w \in DTEST \\ d_{manual}(w) = d_{parser}(w)}} O_w \text{ , and } C = \sum_{w \in DTEST} O_w$$

Skillfulness is thus an analogy to *accuracy*; just each occurrence of a word is weighed by the difficulty of finding its dependency relation.

Now how should the difficulty O_w be defined? We assume that the number of attachments the parser has to choose from increases with the increasing length of the sentence. The proportion is not linear because words prefer to attach to their neighbors and the parser knows that. We want the difficulty be expressed by a 0..1 number, whereas 0 would apply to nothing-to-do cases, and 1 would correspond to an infinite number of possibilities. One of the functions satisfying all mentioned conditions is the following:

$O_w = 1 - \frac{1}{N_w}$, N_w being the number of words in the sentence the word *w* occurs in.

Of course, fancier functions could be designed. It is not clear, how quickly the difficulty should increase with respect to the sentence length. The above definition gives a weight of $^2/_3$ to sentences as short as 3 words; it effectively penalizes only *very short* sentences.

The parser skillfulness on PDT 1.0 d-test data given the above definition of decision difficulty is **74.4 %**.

13.4 Precision and recall

If the parser proposes one and only one dependency for each word the precision and recall in their usual senses will be the same (and equal to what we call accuracy).

We might however want to exploit the parser's ability to assign weights to dependencies and dependency structures. Than we could arrive at structures where a node could have no parent, or several alternative parents. For the purpose of evaluating such structures we define three measures:

Precision *P* is the number of correctly proposed dependencies divided by the number of proposed dependencies.

Recall *R* is the number of correctly proposed dependencies divided by the number of words in the training data.

F-measure is a usual way of getting one number from *P* and *R*. It is close to the mean of *P* and *R* if the two are close to each other. If there is a substantial gap between *P* and *R*, *F* tends to stick with the lower one. F-measure is defined as follows:

$$F = \frac{2 \times P \times R}{P + R}$$

Basically there are two ways of breaking the one-dep-per-word rule:

1. We might require that the parser relax from proposing a dependency whenever the *best* solution is not *good enough*.[51]
2. Or we might request providing two or more solutions (with weights) whenever the *best* solution is not *sufficiently better* than the next one in the row.[52]

[51] An empirical threshold would define what *enough* means.
[52] An empirical threshold would define what *sufficiently* means.

Both approaches lead to parses that do not fulfill our constraints on dependency structures (see the Section 4.2). In (1.) the only change is that some nodes have no parents. In (2.) however, any node is now allowed to have zero to *N* parents where *N* is the number of words in the sentence. This is a much more complicated case. It is not clear for instance, how to ensure that a finally selected subset of dependencies does not contain cycles.

The Model Two parser is not prepared to produce alternative dependencies either. It would not know how to check projectivity, nor would it be able to check treeness. Just to get some outline of the usefulness of the weights we ran the following experiment:

Setting 1: No dependency that has been seen less than ten times in the training data can be proposed. Technically we will add it to the tree for the time of parsing but we will remove it after the parser will have finished.

Setting 2: If a dependency wins a round with weight w_1, and there is a dependency of the same dependent node on another governing node, weighed w_2, and $w_2 \geq 0.9\, w_1$, we add the first dependency to the tree and remember the latter as an alternative dependency. All projectivity and other checks will be done against the first dependency only. Furthermore, if there is another dependency, weighed w_3, and $w_3 \geq 0.9\, w_2$, we remember it as well, and so on recursively. We do not take the weights into consideration when computing precision and recall.

The following table summarizes *P*, *R*, and *F* for experiments 1 and 2. Even though there is a large space of other possibilities that could be explored, we believe that this is a useful (and the first one published for Czech) evaluation of providing alternative dependency structures.

Setting	**Precision**	**Recall**	**F-measure**
Model Two	74.7	74.7	74.7
Setting 1	78.3	66.6	72.0
Setting 2	71.6	75.9	73.7

13.5 Seriousness of errors

If we look at the errors made by the parser we realize that some errors are less serious than others. For instance, sometimes the parser does not succeed to reproduce the structure of nested coordination, although most coordination relations are preserved.

To take a more general view, to what extent is it important to render all commas correctly? Intuitively, all errors in attaching nodes likely to disappear on the tectogrammatical layer are less significant. Most commas belong to that class.

The manually annotated s-tags present in the test data provide some hint as to the syntactic importance of nodes. Most of the not-so-important nodes bear an s-tag beginning with Aux. However, we must be careful. AuxP nodes are very important: this s-tag denotes a preposition, thus hanging AuxP corresponds to the preposition attachment problem. Similar are AuxC nodes (subordinative conjunctions), and AuxT (reflexive particles) are also important.

The following table presents separate accuracy figures for attachments of words with different s-tags (as found in manual annotation of test data):

S-tag(s)	**Accuracy**	
Pred, Pred_Pa	77.9	81.8
Sb	81.9	
Obj, ObjAtr, Obj_Pa	83.1	
Pnom	83.7	
Adv, AdvAtr, Adv_Pa	88.1	
AuxP	67.2	
AuxC	60.1	
Atr, AtrAdv, AtrAdv_Pa, AtrAtr, AtrObj	87.1	
Atv, AtvV, AtvV_Pa	40.0	
*_Ap	30.0	42.7
*_Co	47.5	
Apos, Apos_Pa	30.4	
Coord, Coord_Pa	35.6	
AuxR, AuxT, AuxV	87.4	74.2
AuxG, AuxO, AuxX, AuxY, AuxY_Pa, AuxZ, AuxZ_Pa	57.4	
AuxK, AuxK_Pa	96.8	
ExD, ExD_Pa	68.7	

There are also some features of the PDT annotation scheme that – if not even inconsistent – are unfriendly to statistical parsers. Parsing errors in nodes that are subject to such features would also be hot candidates for the label "less significant". For details, see Zeman (2002b).

Finally we checked how the accuracy will change when we eliminate sentences containing some "dangerous" s-tags from both the training and test data. Coordinations and appositions form one such group of s-tags. ExD tags indicate sentences, in which some words lack their parents due to deletions, and may be attached to unusual substitute parents instead.

S-tags banning sentence processing	**Accuracy**	**Sentence accuracy**
none	74.7	21.0
ExD	76.1	16.4
Coord, Apos	82.8	27.9
ExD, Coord, Apos	83.0	36.2

13.6 Accuracy vs. amount of training data

The following table demonstrates how accuracy grows with the increasing size of training data. In the experiments we simply took the first *n* sentences of the standard PDT 1.0 training data. No attempt was made to balance the data in any way.

Training sentences	Training words	Events recorded	File size	Accuracy
1,000	14,080	83,475	1.7 MB	66.8
10,000	151,205	581,834	13 MB	71.5
25,000	411,706	1,417,306	31 MB	73.4
50,000	848,922	2,506,040	55 MB	74.3
73,088	1,255,590	3,448,365	76 MB	74.7

One may also ask what would be the accuracy if we went through whole the training data set but threw away everything seen only once. Or, maybe, not once but *n*-times. We would not spare the necessity of possessing a large enough treebank but once trained, the parser would require less memory and the parsing could proceed faster. The following table compares the changes in statistics file size and in the accuracy with increasing *n*.

Must have been seen	Events recorded	File size	Accuracy
> 0×	3,448,365	76 MB	74.7
> 1×	944,195	20 MB	74.4
> 2×	538,557	11 MB	74.3
> 5×	238,984	5 MB	73.9
> 10×	123,386	2 MB	73.5

13.7 Accuracy vs. type of data

Hajič et al. (1998) observed that it was substantially easier to parse newswire texts of *Lidové noviny*, *Mladá fronta Dnes*, and *Českomoravský Profit*, than the scientific texts in *Vesmír*.[53] One apparent reason was the higher average sentence length in *Vesmír* (cf. Section 13.1). Most likely it was not the only reason, though. The scientific texts tend to use unusual vocabulary, more complicated or bookish structures etc.

Hajič et al. used the PDT 0.5 test data that contained texts from all the sources. PDT 1.0 d-test data, however, consist solely of *Lidové noviny* texts. That is why we cannot provide separate accuracy for scientific texts. Nevertheless the training data contain all sources; so there is a legitimate question: Will the accuracy rise if we only train on data from the same source? Or at least, will it be better than if trained on the same amount but mixed composition of data? The following table shows that for

[53] See also the Section 7.9.1.

comparable training data sizes it is better to use same-source texts but the difference is minimal.

Training data				Accuracy
Source	**Files**	**Sentences**	**Words**	
Lidové noviny	865	39,978	698,671	74.2
Proportionally all	864	39,687	679,665	74.1

So far we have assessed the source and variability of training data. To look at the variability of the test data, we split the data into 100-sentences blocks and evaluated the accuracy of each block separately.

The following observations may shed some light on the problem of selecting test data of appropriate size.[54]

- The accuracy of the worst block is **67.7 %**.
- The accuracy of the best block is **82.2 %**.
- The 12 best blocks (1200 sentences) together have an accuracy of **80.1 %**. Similarly, the 12 worst blocks together have **69.8 %**.
- The parser steps through the data in the alphabetical order of the file names. After the first 1500 sentences the current accuracy was over 77 %. It lasted between 76 and 77 % until the sentence No. 3000, and until No. 4200 the accuracy was still over 75 %. (The total number of test sentences is 7319.)

[54] Do not get confused — of course, for most of this thesis the test data are fixed for the sake of comparability.

14 Improving the state-of-the-art parser: a superparser

Although we have gone a long way in terms of improving parsing accuracy over Zeman (1997), the final Model Two parser still performs almost 10 % below the best-known parsers for Czech. The Collins' parser has been reported (Hajič et al. (1998), Collins et al. (1999)) to achieve 80.0 % on PDT 0.5; we have found that on PDT 1.0, its accuracy rises to 82.5 %. Moreover, Eugene Charniak has adapted his English parser (Charniak (2000)) to Czech in 2002-2003. His accuracy is the best known for Czech at the moment, 84.3 %.[55]

Interestingly enough, the Model Two parser does not make some errors the two parsers just mentioned do (and vice versa, naturally). This fact creates space for combining all three parsers, hoping that the resulting accuracy would be better than any of the parsers can achieve alone.

In this chapter, we first discuss some previous work, especially the possibilities of generating several different parser outputs for a text using just one parser (Section 14.1). Then we summarize the existing parsers (Section 14.2). Finally we describe the combining algorithm and evaluate the results separately for three parsers (Section 14.3), and for all seven parsers (Section 14.4).

14.1 Generating classifiers

A parser can be viewed as a classifier similar to taggers. While a tagger assigns a POS tag to each input word, a parser assigns a dependency (index of another word). There has been an increasing interest in combining classifiers during the past decade. As it is not common to have several different classifiers at disposal, the research concentrated on two issues: 1. How to generate a number of classifiers given a single one we have; 2. How to combine them?

Basically there are two main approaches to the former issue: bagging (Breiman (1994)) and boosting (Freund and Schapire (1996)). Both methods could also be used together in a hybrid system, which could further be combined with using several different-architecture classifiers (parsers).

With **boosting**, a parser is applied to its own training data and weights are assigned to each sentence: the more errors, the higher weight. Then the parser is re-trained, and asked to pay more attention to points with high weights. The process has to be repeated many times; after each training iteration we get a new (newly trained) parser. Finally all parsers are combined by simple voting. A drawback is that hundreds of iterations are typically needed. Given the parsing times of our parser, it would take days to boost it.

Another way of multiplying classifiers is **bagging**. Different parsers can be simulated by training the same parser on different data. It is undesirable to lessen the training data size just because we need several data sets instead of one. Fortunately,

[55] To our knowledge, this result has not been published.

there is a workaround: we can randomly select 75 % of the available data and fill in the remaining 25 % by repeating random sentences from the 75%-part. This technique produces as many *bags* of data as required while keeping the size of all bags close to the size of the original data.

The third possibility is having several classifiers for the same classifying problem that are different enough so they do not make all the same errors. We are fortunate enough to have at least three parsers at disposal, so this is the only direction we are going to investigate later in this chapter.

14.2 Summary of the parsers

Besides Charniak's and Collins' parsers, there are couple of other parsers that have not been published but are applicable to PDT. In the following table we summarize 7 parsers we are able to deploy in our experiments. We will test two sets: all seven parsers, and the subset of dz+ec+mc.

Parser	Author	Brief description	Accuracy
dz	Daniel Zeman	The Model Two parser described in this thesis.	74.7
ec	Eugene Charniak	A maximum-entropy inspired parser, home in constituency-based structures. English version described in Charniak (2000), Czech adaptation 2002-2003, unpublished.	84.3
mc	Michael Collins	Uses a probabilistic context-free grammar, home in constituency-based structures. Described in Hajič et al. (1998) and Collins et al. (1999).	82.5
zž	Zdeněk Žabokrtský	Purely rule-based parser, rules are designed manually, just a few lexical lists are collected from the training data. 2002, unpublished.[56]	75.2
th-pshrt	Tomáš Holan	Three parsers. Two of them use a sort of push-down automata and differ from each other only in the way they process the sentence (left-to-right or right-to-left). Descriptions to appear in Holan (2004).[57]	62.8
th-l2r			69.9
th-r2l			71.7

14.3 Combining three parsers (ec, mc, dz)

If we compare the errors the various parsers make we will realize that 65.9 % of dependencies are correctly assigned by all 3 parsers. That is the lower bound on accuracy of the parser combination. Theoretical accuracy maximum (at least one parser knows the correct answer) is 92.4 % — but the assumption is we have an oracle able to

[56] Zdeněk Žabokrtský can be reached at CKL MFF UK, visit http://ckl.mff.cuni.cz/.
[57] Tomáš Holan can be reached at KSVI MFF UK, visit http://ksvi.mff.cuni.cz/.

always tell which parser is correct. Our requirement is not to sink under 84.3 %, which is the accuracy of the best parser alone.

The following table shows number of times a parser proposed correct dependency while all the others were wrong. Absolute numbers are accompanied by the percentage of test data they correspond to.

Who is correct		**How many times**	
a single parser (all other wrong)	ec	5411	4.3 %
	mc	3458	2.7 %
	dz	2722	2.2 %
all parsers		83080	65.9 %
majority		104815	83.2 %
at least one parser		116406	92.4 %

The table reveals that even the worst parser produces considerable number of correct dependencies the other two parsers are not able to find. This fact supports our hope that the parsers are sufficiently independent to grant them right to vote about the final result. If two parsers produced too similar results, there would be the danger that they push all their errors through, blocking any meaningful opinion of the third parser.

On the other hand, there is little chance that any parser will be able to push through the things it specializes in. It is very difficult to realize that a parser is right if both the others reject its proposal. Later in this chapter we will describe some experiments with using morphological context to detect particular parsers' strong and weak points. However, the real power is in majority of votes.

14.3.1 Balanced and unbalanced context-free voting

The ec parser is known to produce the most-accurate results. Thus it is reasonable to prefer ec's output whenever the three parsers disagree. Only if mc+dz agree on a different solution ec is outvoted.

These parsers agree in 53.2 % on a correct result and in 46.8 % on an error. Thus the accuracy improvement is not dramatic; it is interesting, though: **85.5 %** in contrast to the 84.3 % of the ec parser alone.

Hajič et al. (1998, Chapter 5)[58] applied bagging to PDT 0.5 and Collins parser. They distinguish **balanced** and **unbalanced** combining of parsers. Balanced combinations propose exactly one dependency for each word. Unbalanced combinations retain all dependencies proposed by at least half of the parsers. Precision and recall has to be computed instead of accuracy (see Section 13.4). We have tested both balanced and unbalanced combining.

Note that even balanced voting does not guarantee that the resulting set of dependencies conforms to our definition of a dependency tree structure. That may or may not matter depending on the application that wants to use the parsing results. We believe that given the definition of accuracy (computed on words, not sentences), it is OK

[58] See also Hladká (2000), Chapter 5.

to consider each word separately. If "treeness" is required, further checks would have to be performed and the accuracy improvement would naturally be less impressive. Note however that (to our knowledge) the previous work on parser combination in Hajič et al. (1998) does not care whether the result is a tree, and neither do some of the individual parsers. Henderson and Brill (1999) prove that their combination of constituency-based parsers could not produce crossing brackets; unfortunately, there is no analogy in the dependency framework (treeness of the resulting structure cannot be taken for granted).

14.3.2 Using context

If we have only three parsers we could use context to detect two kinds of situations:

1. If each parser has its own proposal and a parser other than ec shall win.
2. If two parsers agree on a common proposal but even so the third one should win. Most likely the only reasonable instance is that ec wins over mc+dz.

"Context" can be represented by a number of features, starting at morphological tags and ending up at complex queries on structural descriptions. We use the core features the individual parsers themselves train on: the m-tags, in a couple of flavors. Either we look at the m-tag reduced the way described in Section 8.4, or we only look at the part-of-speech and case, or at POS only. We consider the m-tag of the dependent node, and the m-tags of the governors proposed by the individual parsers.

We have to learn the strengths of a parser on some held-out data that have not been used to train any of the parser. Of course, the held-out data must not overlap with the test data either. So we reduce testing on the first 76 test files only (last file included is lv28) while putting the rest aside as the held-out data.[59] Here are the accuracies of the individual parsers on the new test data subset:

Parser	Accuracy
ec	85.0
mc	83.3
zž	76.2
dz	75.5
th-r2l	72.3
th-l2r	70.3
th-pshrt	63.5

[59] We spare by that a lot of effort with re-training all the parsers on a subset of the training data and re-running them on the held-out data. In fact, for the purpose of this research we do not need to run the foreign parsers at all. We only work with their outputs on PDT 1.0 d-test data, kindly provided by the respective parsers' authors. Of course, an end-user-ready superparser would only run after we have brought all the individual parsers to life.

The next table presents recomputed comparison of ec, mc and dz on the new test data subset:

Who is correct	**How many times**	
all parsers	42090	67.2 %
at least one parser	58126	92.7 %

Henderson and Brill (1999) experimented with combining constituency-based parsers for English. They were able to improve the F-score on the Penn TreeBank (91.3%, as compared to 87.7% of their best parser).

They report that context did not help them to outperform simple voting. The results of our experiments, summarized in the following table, seem to support their hypothesis. Although we observed some positive influence of the context, the improvement was very small.

Combination method	**Accuracy**	
	PDT 1.0 d-test	subset
ec only (baseline)	84.3	85.0
balanced: absolute majority, or ec	85.5	86.2
context: m-tag of dep. node and ec-proposed gov. node; in specific contexts wins over mc+dz; otherwise majority, or again ec if there is no majority	NA	86.3

Finally we compare the balanced and unbalanced methods. The unbalanced combination of the three parsers achieves **the best parsing result of this thesis,** and the best result published for Czech/PDT so far.

Method	**Precision**	**Recall**	**F-measure**
ec only (baseline)	85.0	85.0	85.0
balanced context-sensitive combination	86.3	86.3	86.3
unbalanced context-free combination	**89.5**	84.0	**86.7**

Some of the contexts in which ec outperformed the common opinion of mc+dz follow.

Tag of dependent	**Tag of governor proposed by ec**	**No. of times ec was right**	**Context occurrences**	**Percent cases ec was right**
J^	#	44	67	65.7
Vp	J^	28	53	52.8
VB	J^	26	46	56.5
N1	Z,	21	38	55.3
Rv-1	Vp	13	25	52.0
Z,	Z,	8	15	53.3

A1	N1	8	15	53.3
Vje	J^	9	14	64.3
N4	Vf	9	12	75.0

14.4 Combining all parsers

First of all let us again present the brief comparison of the parsers' errors, weak and strong points. The hypothetical upper bound of accuracy achievable by voting has risen to almost 96 %. However, the results of experiments bring less improvement than the combination of three parsers discussed above.

Who is correct		**How many times**	
a single parser (all other wrong)	ec	1976	1.6 %
	zž	1388	1.1 %
	mc	1081	0.9 %
	thr2l	500	0.4 %
	thpshrt	499	0.4 %
	dz	466	0.4 %
	thl2r	395	0.3 %
all parsers		54732	43.4 %
at least half of the parsers		97913	77.7 %
majority or ec		106768	84.7 %
at least half of the parsers, or ec while there is no absolute majority		107213	85.1 %
absolute majority, or ec+2, or mc+2, or ec		107460	**85.3 %**
at least one parser		120917	95.9 %

Balanced vs. unbalanced combination:

Method	**Precision**	**Recall**	**F-measure**
ec only (baseline)	85.0	85.0	85.0
balanced context-free voting	85.8	85.8	85.8
unbalanced context-free combination	**90.7**	78.6	84.2

15 Related work

Seven current parsers for Czech have been introduced in Chapter 14. Please refer to that chapter for comparison of their performance. Here we only compile some additional works that have not been mentioned yet.

The research on Czech parsing can be traced back to the RUSLAN system (Oliva (1989)) that was once part of a Czech-Russian machine translation system. It was implemented in Systemes Q. No evaluation on PDT data is available but it would probably be quite bad, as RUSLAN's coverage was tailored to a limited domain.

Holan, Kuboň, Oliva and Plátek (see Holan et al. (1998), Kuboň et al. (2001), Holan (2001), Kuboň (2001)) have published a line of papers on problems of Czech robust parsing, grammar checking, theoretical issues of non-projectivity, computational complexity of parsing free-word-order languages etc. To my knowledge, no evaluation of their proposals on PDT is available, as they focus mainly on the problematic phenomena in language. Note that the Holan's parsers discussed in Chapter 14 are not directly related to this work.

Smrž and Horák (1999, 2000) present a grammar-based chart-parser for Czech. However, their system cannot be compared to ours in terms of accuracy, as they do not publish any accuracy (or precision/recall) evaluation; they focus just on the speed of the parser. In Horák and Smrž (2001, 2002) they report a precision of 79.3 %. Unfortunately that result is not comparable to ours either. The papers do not specify on which data and how the precision was measured but we asked the authors directly. There are substantial differences in their and our methodology, namely:

- They test whether whole trees match, not just dependencies.
- Their "precision" is defined as relative frequency of cases when the correct parse is present among all parses in the output.
- They "estimate" the precision on 100 sentences, manually annotated with constituency-based (as opposed to dependency-based) syntactic structures. Evaluation on PDT 1.0 test data may give different results.

Ribarov (2004) has recently published results and qualitative analyses of several other Czech parsers, mainly based in transformational rules. His "naïve" parsers achieved around 70% accuracy on PDT d-test data. The accuracy of his best parser (perceptron-based) is close to 72%.

The group of not-yet-published results (see zž and th* parsers in Chapter 14) would not be complete without mentioning Honetschläger's[60] work. His principal idea is to take a window of *d* words in the text. Based on the contents of the window he proposes a dependency between two words in the window — that is one rule. He repeatedly scans the training data for rules that never (or almost never) make an error; out of such, the most frequently applicable rules take preference. There are also rules describing exceptions for regular rules, and a simple statistical model for attaching words not covered by any rule. The accuracy of Honetschläger's model is reportedly 73.6 %.

[60] Václav Honetschläger can be reached at CKL, visit http://ckl.mff.cuni.cz/.

Schwartz, Chelba and Jelinek (see Chelba and Jelinek (1998), Hajič et al. (1998)) tried to adapt the parser in Chelba+Jelinek's *structured language model* for Czech. They achieved an accuracy of 68.2 % on PDT 0.5 d-test data.

Jones and Kuo (see Hajič et al. (1998)) proposed a "parsing by translation" approach. Their idea was that we have good parsers for English and there are Czech and English treebanks. If the Czech sentence could be translated into English and parsed there, the tree structure could then be transferred back to Czech, using analogy. The theoretical discussion has never proceeded to coding. Today, the opposite way is taken when parsing helps machine translation and transferring parsed structures give better results than transferring plain texts (see Cuřín et al. (2002), Čmejrek et al. (2003)).

Last but not least there are some considerable contributions to partial parsing of Czech. Let us name two of them. Lopatková (2001) investigates homonymy of prepositional phrases in Czech. Žáčková (2002) proposes a partial parsing system implemented in Prolog, able to find simple noun phrases, prepositional phrases, and – most interestingly – some discontinuous verbal groups.

16 Conclusion

We have presented a method for finding surface shape of the sentence shallow structure – a process we call dependency parsing. We have documented a number of features that may be modeled statistically, by which some of them help to improve the accuracy of the parsing. As a side effect, we made and described many quantitative observations about the Prague Dependency Treebank.

A significant improvement over the early experiments of Zeman (1997) has been demonstrated. Although our parser has not achieved the state-of-the-art accuracy, we believe that the present work brings more than just a "well documented failure", for the following reasons:

1. We have shown that the presented parser can cooperate with the better parsers so that the resulting super-parser improves over all of them.
2. There is still space for future work (see below). Unfortunately a complete investigation of that space is a long-term goal and thus could not be encompassed in the present work.

16.1 Future work

It seems that we have reached the limits of the Model Two. Every new promile of accuracy has to be strenuously fought out and, more seriously, paid for by a manual interference in the statistical core. That of course decreases portability to new domains and languages, which is the usually proclaimed merit of statistical methods.

However, we think that a new *Model Three* can be built that will overcome the apparent advantage of constituency-based parsers: the knowledge of sister nodes, in addition to parent-child relations. More generally, such a model could take into account all important tree fragments in a similar way to what Data-Oriented Parsers (DOPs)[61] do.

If that is possible, we believe that the state-of-the-art accuracy can be beaten. Moreover, the model would come out from the current work and would be housed in the dependency framework. It would thus be more straightforward than the current state-of-the-art parsers, as they require translating the data into an immediate-constituency framework.

Integrating syntactic parsing with morphological tagging provides another chance. On several occasions we complained that erroneous tags misled the parser. It is well known that some tagging errors can be avoided when at least a rough syntactic analysis is available — so if the parser can do the tagging and parsing in parallel it shall gain better trees.

[61] See Bod et al. (2003).

17 Index

18 References

Bod et al. 2003
Rens Bod, Remko Scha, Khalil Sima'an: *Data-Oriented Parsing.* CSLI Studies in Computational Linguistics (ed. by Ann Copestake). Center for the Study of Language and Information, Leland Stanford Junior University, Stanford, California, 2003

Böhmová et al. 2000
Alena Böhmová, Jan Hajič, Eva Hajičová, Barbora Hladká: *The Prague Dependency Treebank: Three Level Annotation Scenario.* In: Anne Abeillé (ed.): Treebanks: Building and Using Syntactically Annotated Corpora. http://ufal.mff.cuni.cz/pdt/ Kluwer Academic Publishers, Dordrecht, Netherlands, 2000

Bojar 2002
Ondřej Bojar: *Automatizovaná extrakce lexikálně syntaktických údajů z korpusu (diplomová práce).* Univerzita Karlova, Praha, Czechia, 2002

Bloomfield 1933
Leonard Bloomfield: *Language*. 564 pp. Holt, New York, New York, 1933

Breiman 1994
Leo Breiman: *Bagging Predictors.* http://www.work.caltech.edu/cs156/01/papers/bagging.ps.gz Technical Report 421, Department of Statistics, University of California at Berkeley, Berkeley, California, 1994

Brent 1991
Michael Brent: *Automatic acquisition of subcategorization frames from untagged text.* In: Proceedings of the 29th meeting of the ACL, pp. 209–214. Berkeley, California, 1991

Brent 1993
Michael Brent: *From grammar to lexicon: unsupervised learning of lexical syntax.* In: Computational Linguistics, vol. 19, no. 3, pp. 243–262. Association for Computational Linguistics, New Brunswick, New Jersey, 1993

Brent 1994
Michael Brent: *Acquisition of subcategorization frames using aggregated evidence from local syntactic cues.* In: Lingua, vol. 92, pp. 433–470, reprinted in: Acquisition of the Lexicon, L. Gleitman and B. Landau (eds.). MIT Press, Cambridge, Massachusetts, 1994

Carroll and Minnen 1998
John Carroll, Guido Minnen: *Can Subcategorization Probabilities Help a Statistical Parser?* In: Proceedings of the 6th ACL/SIGDAT Workshop on Very Large Corpora (WVLC-6). Université de Montréal, Montréal, Québec, 1998

Carroll and Rooth 1998
Glenn Carroll, Mats Rooth: *Valence Induction with Head-Lexicalized PCFG.* Proceedings of the 3rd Conference on Empirical Methods in Natural Language Processing (EMNLP-3). Granada, Spain, 1998

Charniak 1997
Eugene Charniak: *Statistical Techniques for Natural Language Parsing.* In: AI Magazine, vol. 18(4), pp. 33–44. ISSN 0738-4602. American Association for Artificial Intelligence, Menlo Park, California, 1997

Charniak 2000
Eugene Charniak: *A Maximum-Entropy-Inspired Parser.* In: Proceedings of NAACL. Seattle, Washington, 2000

Chelba and Jelinek 1998
Ciprian Chelba, Frederick Jelinek: *Exploiting Syntactic Structure for Language Modeling.* Proceedings of the 36th Annual Meeting of the ACL and the 17th International Conference on Computational Linguistics (COLING-ACL 98). Université de Montréal, Montréal, Quebec, 1998

Chomsky 1957
Noam Chomsky: *Syntactic Structures.* Mouton & Co. Publishers, Den Haag, Netherlands, 1957

Čmejrek et al. 2003
Martin Čmejrek, Jan Cuřín, Jiří Havelka: *Czech-English Dependency-based Machine Translation.* In: EACL 2003 Proceedings of the Conference, pp. 83–90. Budapest, Hungary, 2003

Collins 1999
Michael Collins: *Head-Driven Statistical Models for Natural Language Parsing (PhD thesis)*. http://www.research.att.com/~mcollins/ University of Pennsylvania, Philadelphia, Pennsylvania, 1999

Collins et al. 1999
Michael Collins, Jan Hajič, Eric Brill, Lance Ramshaw, Christoph Tillmann: *A Statistical Parser of Czech.* In: Proceedings of the 37th Meeting of the ACL, pp. 505–512. University of Maryland, College Park, Maryland, 1999

Cuřín et al. 2002
Jan Cuřín, Martin Čmejrek, Jiří Havelka: *Czech-English Dependency-based Machine Translation.* In: Prague Bulletin of Mathematical Linguistics, vol. 78, pp. 103 – 118. Univerzita Karlova, Praha, Czechia, 2002

Freund and Schapire 1996
Yoav Freund, Robert E. Schapire: *Experiments with a New Boosting Algorithm.* In: Proceedings of the 13th International Conference on Machine Learning, pp. 148–156. Morgan Kaufmann. Bari, Italy, 1996

Hajič 1998
Jan Hajič: *Building a Syntactically Annotated Corpus: The Prague Dependency Treebank.* In: Issues of Valency and Meaning, pp. 106–132. Karolinum, Praha, Czechia, 1998

Hajič 2004
Jan Hajič: *Disambiguation of Rich Inflection (Computational Morphology of Czech).* http://nlp.cs.jhu.edu/~hajic/morph.html Karolinum, Praha, Czechia, 2004

Hajič and Hladká 1998
Jan Hajič, Barbora Hladká: *Tagging Inflective Languages: Prediction of Morphological Categories for a Rich, Structured Tagset.* In: Proceedings of the 36th Annual Meeting of the ACL and the 17th International Conference on Computational Linguistics (COLING-ACL 98), pp. 483–490. Université de Montréal, Montréal, Québec, 1998

Hajič et al. 1998
Jan Hajič, Eric Brill, Michael Collins, Barbora Hladká, Douglas Jones, Cynthia Kuo, Lance Ramshaw, Oren Schwartz, Christoph Tillmann, Daniel Zeman: *Core Natural Language Processing Technology Applicable to Multiple Languages. The Workshop 98 Final Report.* http://www.clsp.jhu.edu/ws98/projects/nlp/report/ Johns Hopkins University, Baltimore, Maryland, 1998

Hajič et al. 1999
Jan Hajič, Jarmila Panevová, Eva Buráňová, Zdeňka Urešová, Alla Bémová: *Annotations at Analytical Level (English translation by Zdeněk Kirschner)* http://ufal.mff.cuni.cz/pdt/Corpora/PDT_1.0/Doc/aman-en/index.html Univerzita Karlova, Praha, Czechia, 1999

Hajič et al. 2001
Jan Hajič, Pavel Krbec, Pavel Květoň, Karel Oliva, Vladimír Petkevič: *Serial Combination of Rules and Statistics: A Case Study in Czech Tagging.* Proceedings of the 39th Annual Meeting of the ACL (ACL-EACL 2001). Université de Sciences Sociales, Toulouse, France, 2001

Hajič et al. 2001b
Jan Hajič, Eva Hajičová, Petr Pajas, Jarmila Panevová, Petr Sgall, Barbora Vidová Hladká: *The Prague Dependency Treebank.* CD-ROM LDC2001T10. http://ufal.mff.cuni.cz/pdt/ Linguistic Data Consortium, University of Pennsylvania, Philadelphia, Pennsylvania, 2001

Hajič et al. 2003
Jan Hajič, Jarmila Panevová, Zdeňka Urešová, Alla Bémová, Veronika Kolářová, Petr Pajas: *PDT-VALLEX: Creating a Large-coverage Valency Lexicon for Treebank Annotation.* In: Joakim Nivre, Erhard Hinrichs (eds.): Proceedings of The Second Workshop on Treebanks and Linguistic Theories, pp. 57–68. Växjö University Press, Växjö, Sweden, 2003

Hajičová et al. 2004
Eva Hajičová, Jiří Havelka, Petr Sgall, Kateřina Veselá, Daniel Zeman: *Issues of Projectivity in the Prague Dependency Treebank.* In: Prague Bulletin of Mathematical Linguistics, vol. 81. Univerzita Karlova, Praha, Czechia, 2004

Henderson and Brill 1999
John C. Henderson, Eric Brill: *Exploiting Diversity in Natural Language Processing: Combining Parsers.* In: Proceedings of the Fourth Conference on Empirical Methods in Natural Language Processing (EMNLP-99), pp. 187–194. http://research.microsoft.com/~brill/Pubs/pcomb.ps University of Maryland, College Park, Maryland, 1999

Hladká 2000
Barbora Hladká: *Czech Language Tagging (PhD thesis)* http://ckl.mff.cuni.cz/~hladka/phd_page.html Univerzita Karlova, Praha, Czechia, 2000

Holan 2001
Tomáš Holan: *Nástroje pro vývoj závislostních analyzátorů přirozených jazyků s volným slovosledem (PhD thesis).* Univerzita Karlova, Praha, Czechia, 2001

Holan 2003
Tomáš Holan: *K syntaktické analýze českých (!) vět.* In: MIS 2003 Josefův Důl, Sborník semináře, David Obdržálek, Jana Tesková (eds.), pp. 66–74. Matfyzpress, Praha, Czechia, 2003

Holan 2004
Tomáš Holan: *Tvorba závislostního syntaktického analyzátoru.* In: MIS 2004 Josefův Důl, Sborník semináře, David Obdržálek, Jana Tesková (eds.). Matfyzpress, Praha, Czechia, in print

Holan et al. 1998
Tomáš Holan, Vladislav Kuboň, Karel Oliva, Martin Plátek: *Two Useful Measures of Word-Order Complexity.* In: A. Polguere, S. Kahane (eds.): Proceedings of the Coling'98 Workshop on Processing of Dependency-Based Grammars, pp. 21–28. Université de Montréal, Montréal, Québec, 1998

Horák and Smrž 2001
Aleš Horák, Pavel Smrž: *Efficient Sentence Parsing with Language Specific Features: A Case Study of Czech.* In: Proceedings of the IWPT'2001 (the 7th International Workshop on Parsing Technologies), pp. 221–224. http://nlp.fi.muni.cz/publications/iwpt2001_hales_smrz/iwpt2001_hales_smrz.pdf Běijīng Dàxué / Qīnghuá Dàxué Chūbānshè, Běijīng, China, 2001

Horák and Smrž 2002
Aleš Horák, Pavel Smrž: *Best Analysis Selection in Inflectional Languages.* In: Proceedings of the 19th International Conference on Computational Linguistics (COLING 2002), pp. 363–368. http://nlp.fi.muni.cz/publications/coling2002_hales_smrz/coling2002_hales_smrz.pdf Zhōngyāng Yánjiùyuàn, Táiběi, Taiwan, 2002

Hudson 1984
Richard Hudson: *Word Grammar.* Blackwell, Oxford, Britain, 1984

Kuboň 2001
Vladislav Kuboň: *Problems of Robust Parsing of Czech (PhD thesis).* Univerzita Karlova, Praha, Czechia, 2001

Kuboň et al. 2001
Vladislav Kuboň, Tomáš Holan, Karel Oliva, Martin Plátek: *Word-Order Relaxations & Restrictions within a Dependency Grammar.* In: Proceedings of IWPT, pp. 237–240. 清华大学出版社，北京, China, 2001

Lopatková 2001
Markéta Lopatková: *Homonymie předložkových skupin v češtině a možnost jejich automatického zpracování (PhD thesis)*. Univerzita Karlova, Praha, Czechia, 2001

Manning and Schütze 1999
Christopher D. Manning, Hinrich Schütze: *Foundations of Statistical Natural Language Processing.* ISBN 0-262-13360-1. MIT Press, Cambridge, Massachusetts, 1999

Marcus 1965
Solomon Marcus: *Sur la notion de projectivité.* In: Zeitschrift für mathematische Logik und Grundlagen der Mathematik, vol. 11, pp. 181–192, ISSN 0044-3050. Leipzig, Germany, 1965

Marcus et al. 1993
Mitchell Marcus, Beatrice Santorini, Mary Ann Marcinkiewicz: *Building a Large Annotated Corpus of English: The Penn Treebank.* Computational Linguistics, vol. 19, pp. 313–330. Reprinted in Susan Armstrong (ed.), 1994, Using large corpora, pp. 273–290. MIT Press, Cambridge, Massachusetts, 1993

Nenadić and Vitas 1998
Goran Nenadić, Duško Vitas: *Using Local Grammars for Agreement Modeling in Highly Inflective Languages.* In: Petr Sojka, Václav Matoušek, Karel Pala, Ivan Kopeček (eds.): Text, Speech and Dialogue, Proceedings of the 1st International Workshop, pp. 91–96. Masarykova Univerzita, Brno, Czechia, 1998

Nenadić 2000
Goran Nenadić: *Local Grammars and Parsing Coordination of Nouns in Serbo-Croatian.* In: Petr Sojka, Ivan Kopeček, Karel Pala (eds.): Text, Speech and Dialogue, Proceedings of the 3rd International Workshop, pp. 57–62. Springer LNAI 1902, Brno, Czechia, 2000

Oliva 1989
Karel Oliva: *A Parser for Czech Implemented in Systems Q*. In: Explizite Beschreibung der Sprache und automatische Textbearbeitung XVI. Univerzita Karlova, Praha, Czechia, 1989

Ondruška 2004
Roman Ondruška: *Extrakce informace ze syntakticky anotovaných korpusů (disertační práce).* Univerzita Karlova, Praha, Czechia, 2004

Petkevič 1999
Vladimír Petkevič: *Czech Translation of G. Orwell's '1984': Morphology and Syntactic Patterns in the Corpus.* In: Václav Matoušek, Pavel Mautner, Jana Ocelíková, Petr Sojka (eds.): Text, Speech and Dialogue, Proceedings of the 3rd International Workshop, pp. 77–82. Springer LNAI 1692, Mariánské Lázně, Czechia, 1999

Plátek and Sgall 1978
Martin Plátek, Petr Sgall: *A scale of context sensitive languages: Applications to natural language.* In: Information and Control, vol. 38, pp. 1–20. ISSN 0019-9958. Elsevier, Amsterdam, Netherlands, 1978

Ribarov 2004
Kiril Ribarov: *Automatic Building of a Dependency Tree — the rule-based approach and beyond (PhD thesis).* Univerzita Karlova, Praha, Czechia 2004

Sarkar and Zeman 2000
Anoop Sarkar, Daniel Zeman: *Learning Verb Subcategorization from Corpora: Counting Frame Subsets.* In: Proceedings of the second International Conference on Language Resources and Evaluation (LREC 2000). http://ckl.mff.cuni.cz/~zeman/publikace/2000-08/coling2000.ps Universität des Saarlandes, Saarbrücken, Germany, 2000

Sgall et al. 1986
Petr Sgall, Eva Hajičová, Jarmila Panevová: *The Meaning of the Sentence in its Semantic and Pragmatic Aspects.* Jacob Mey (ed.), 364 pp. ISBN 90-277-1838-5. D. Reidel Publishing Company, Dordrecht, Netherlands, 1986

Siegel 1997
Eric V. Siegel: *Learning Methods for Combining Linguistic Indicators to Classify Verbs.* In: Proceedings of EMNLP-97, pp. 156–162. Providence, Rhode Island, 1997

Smrž and Horák 1999
Pavel Smrž, Aleš Horák: *Implementation of efficient and portable parser for Czech.* In: Text, Speech and Dialogue: Proceedings of the Second International Workshop TSD'1999. Springer LNCS, vol. 1692. Západočeská univerzita, Plzeň, Czechia, 1999

Smrž and Horák 2000
Pavel Smrž, Aleš Horák: *Large Scale Parsing of Czech*. In: Proceedings of Efficiency in Large-Scale Parsing Systems Workshop, COLING 2000, pp. 43–50. ISBN 1-55860-717-X. http://nlp.fi.muni.cz/publications/elsps2000_smrz_hales/elsps2000_smrz_hales.pdf Universität des Saarlandes, Saarbrücken / Luxembourg, 2000

Uhlířová 1967
Ludmila Uhlířová: *On the non-projective constructions in Czech.* In: Prague Studies in Mathematical Linguistics, vol. 3 (1972), pp. 171–181. Academia, Praha, Czechia, 1972

Veselá et al. 2004
Kateřina Veselá, Jiří Havelka, Eva Hajičová: *Condition of Projectivity in the Underlying Dependency Structures*. In: Proceedings of the 20th International Conference on Computational Linguistics (COLING 2004). Université de Genève, Genève, Switzerland, 2004

Viterbi 1967
Andrew J. Viterbi: *Error Bounds for Convolutional Codes and Asymptotically Optimum Decoding Algorithm.* In: IEEE Transactions on Information Theory IT-13: 1260-269. 1967

Žabokrtský and Lopatková 2004
Zdeněk Žabokrtský, Markéta Lopatková: *Valency Frames of Czech Verbs in VALLEX 1.0.* In: Adam Meyers (ed.): Frontiers in Corpus Annotation. Proceedings of the Workshop of the HLT/NAACL Conference, pp. 70–77. http://ufal.mff.cuni.cz/publications/year2004/vallex-naacl04.ps Boston, Massachusetts, 2004

Žáčková 2002
Eva Žáčková: *Parciální syntaktická analýza (češtiny) (PhD thesis)*. Masarykova univerzita, Brno, Czechia, 2002

Zeman 1997
Daniel Zeman: *Pravděpodobnostní model významových zápisů vět.* http://ckl.mff.cuni.cz/~zeman/publikace/diplomka/diplomka.html Univerzita Karlova, Praha, Czechia, 1997

Zeman 1998
Daniel Zeman: *A Statistical Approach to Parsing of Czech.* In: Prague Bulletin of Mathematical Linguistics, vol. 69, pp. 29–37. http://ckl.mff.cuni.cz/~zeman/publikace/1998-06/pbml.html Univerzita Karlova, Praha, Czechia, 1998

Zeman 2001a
Daniel Zeman: *Parsing with Regular Expressions: A Minute to Learn, a Lifetime to Master.* In: Prague Bulletin of Mathematical Linguistics, vol. 75, pp. 29-37. http://ckl.mff.cuni.cz/~zeman/publikace/2001-01/pbml2001.html Univerzita Karlova, Praha, Czechia, 2001

Zeman 2001b
Daniel Zeman: *How Much Will a RE-based Preprocessor Help a Statistical Parser?* In: Proceedings of the 7th International Workshop on Parsing Technologies (IWPT 2001), ISBN 7-302-04925-4. http://ckl.mff.cuni.cz/~zeman/publikace/2001-02/iwpt2001.html Běijīng Dàxué / Qīnghuá Dàxué Chūbǎnshè, Běijīng, China, 2001

Zeman 2002a
Daniel Zeman: *Can Subcategorization Help a Statistical Dependency Parser?* In: Proceedings of the 19th International Conference on Computational Linguistics (COLING 2002). http://ckl.mff.cuni.cz/~zeman/publikace/2002-01/coling-taibei.html Zhōngyāng Yánjiùyuàn, Táiběi, Taiwan, 2002

Zeman 2002b
Daniel Zeman: *How to Decrease Performance of a Statistical Parser*. In: Prague Bulletin of Mathematical Linguistics, vol. 78, pp. 53-62. http://ckl.mff.cuni.cz/~zeman/publikace/2002-02/PBML-2002-09.html Univerzita Karlova, Praha, Czechia, 2002.

Zeman and Sarkar 2000
Daniel Zeman, Anoop Sarkar: *Automatic Extraction of Subcategorization Frames for Czech.* Proceedings of the 18th International Conference on Computational Linguistics (COLING 2000). http://ckl.mff.cuni.cz/~zeman/publikace/2000-06/atensubc.ps European Language Resources Association (ELRA), Athîna, Greece, 2000