Core Natural Language Processing Technology Applicable to Multiple Languages

Workshop '98

Center for Language and Speech Processing
Johns Hopkins University
Baltimore, MD, USA

Final Report

Jan Hajič, Eric Brill, Michael Collins, Barbora Hladká, Douglas Jones, Cynthia Kuo, Lance Ramshaw, Oren Schwartz, Christoph Tillmann, Daniel Zeman

Introduction

Parsing natural languages is a task approached by many people in the past and present and undoubtedly also in the future. It is widely believed that if one can obtain automatically a good parse for a given input sentence (i.e. functions of the words and relations between them), then all the "language understanding" tasks can profit from it.

English has been studied most for this purpose and many good results have been obtained, especially using statistically-based techniques coupled with automatic learning from large annotated corpora. However, these techniques have not been applied to other languages than English very much, especially to languages which display substantially different behavior (both morphologically and syntactically) than English (such as Slavic languages, spoken mostly in Europe by approx. 350 million people).

The Task

Czech has been selected as the "example" language for the Workshop'98 parsing task mainly for two reasons: (1) it is substantially different from English (highly inflectional morphology, free word order), and (2) at least some resources (data, mainly the so-called Prague Dependency Treebank (PDT), and tools, such as morpholgical dictionary and taggers) exist for automatic learning techniques to work reasonably.

We have worked on the following parsers before and during the workshop: Michael Collins' state-of-the-art parser for English (which was being adapted for Czech), Dan Zeman's "direct" dependency parser for Czech, and Ciprian Chelba's and Fred Jelinek's Structured Language Model (as a parser). Also, we were exploring the possibility (and discovering the difficulties) of parsing an "unknown" language using the "Parsing by translation" approach.

On top of those parsers, we were working on the "Classifier Combination" method for combining the results of several parsers to obtain still better results. Although we could not test this method on really different parsers, interesting results have been obtained using various variants of a single parser.

Evaluation Criteria

In order to compare the results among different parsers, we have set up an evaluation scheme which would be applicable to all parsers. As the test data consist of dependency trees, it was natural to choose the following evaluation metric:

An error occurs if and only if a dependency link goes to a wrong governing node.

Every sentence contains one artificial node (tree root), thus the number of nodes representing "real" nodes equals the number of dependency links (counting also links leading to the artificial root node). Parsing accuracy can thus be defined very simply:

Accuracy = number of correct dependency links / number of words in a sentence

provided that each node has been supplied by one and only one upgoing link by the parser being evaluated. Precision and recall can be easily defined, should some parser(s) provide partial parses or multiple parses.

Special evaluation tools have been developed, and all parser "developers" have been instructed to provide the output of their respective parsers in the test data format for automatic evaluation.

There were two test sets of data. One of them (the development test set) has been freely available, and people could in fact use it in any way they wanted. The final evaluation test set has been kept separate for final and completely unbiased evaluation.
Both sets consisted of over 3500 sentences, whereas the training set contained over 19 thousand sentences. The original partitioning of the Prague Dependency Treebank into training and the two test sets is still kept in the freely available version of the PDT (http://ufal.ms.mff.cuni.cz), so that results obtained later and/or by other people are directly comparable.

Data Preparation

The Prague Dependency Treebank (see also ufal.ms.mff.cuni.cz -- then go to Projects, and to PDT) has been prepared for the Workshop with full annotations on level 1 (morphology) and level 2 (dependency syntax). Although the PDT is still being developed (final version shoud be available before the end of 1999), we were able to obtain almost half a million of words annotated on these two levels at the start of the Workshop. Substantial effort has been spent before the workshop and even in the first three weeks of the workshop to check the PDT and merge the annotation levels 1 and 2 into a single data resource, which would otherwise be necessary only sometimes during 1999 when all the manual annotation work is finished.

The sizes of the data:

	Sentences	Words
Training	19126	327597
Development test	3697	63718
Evaluation test	3787	65390

Resources and Tools

The following resources were available for the workshop (part of them has been developed in the past without direct relation to the workshop, but some, such as the evaluation tool and the tree viewer have been developed by the team members berfore and during the workshop):

- the Prague Dependency Treebank (now dubbed "PDT version 0.5", almost 450,000 words, or about 26,000 sentences), with each word annotated with its original textual form, lemma, morphosyntactic tag, analytical (syntactic) function and its governing node. The original position of the word in the input sentence was also preserved.
- Bilingual English-Czech data (2 mil. words) from the Czech edition of Reader's Digest, sentence-aligned, for the "parsing by translation" experiments
- Czech morphological dictionary and morphological analyzer, which for (almost) each word from a Czech text produces a list of possible grammatical meanings (on the morphological level) and all possible lemmas. The coverage of the dictionary is about 98% of a newspaper text. The tagset is very detailed (as the inflectional nature of Czech dictates) and currently contains 3128 tags.
- Czech tagger (so called "exponential" tagger), which can disambiguate the tags as provided by the morphlogical dictionary. The error rate of the tagger is at about 92% (flat, using all morphological categories), but for example for the POS alone it is 99%.

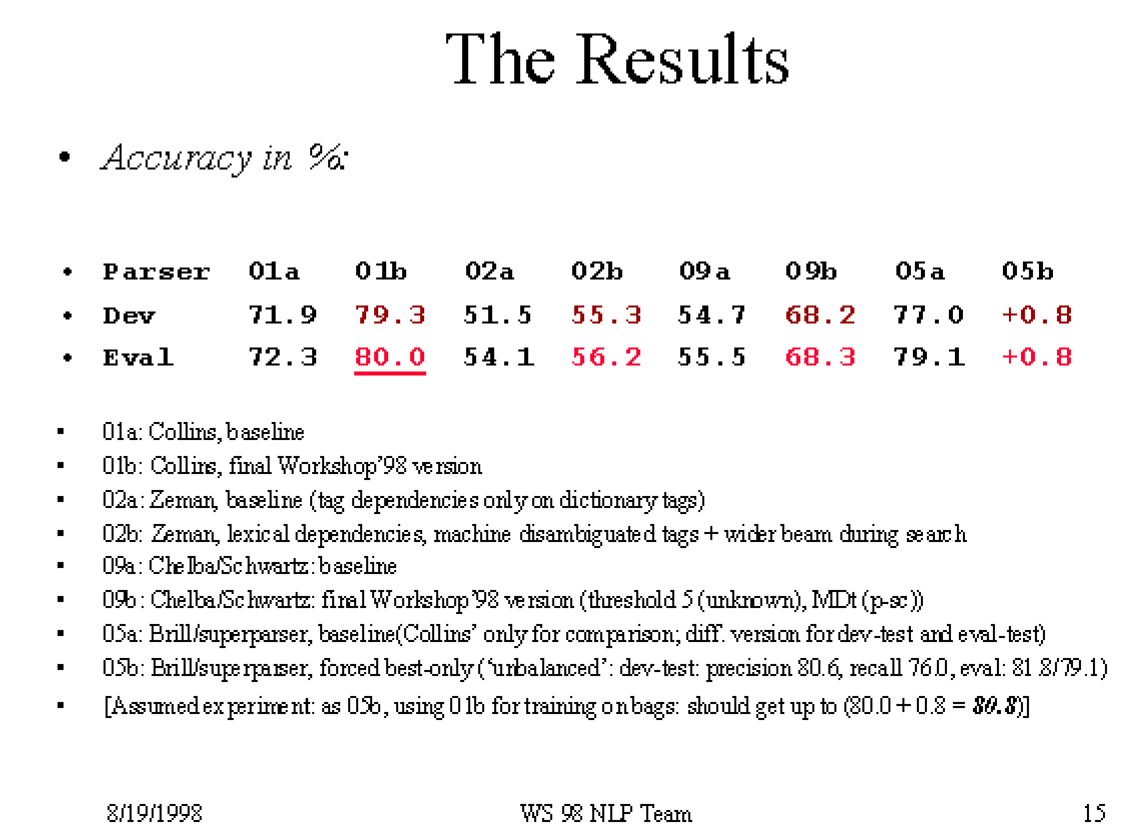
- Evaluation tool for comparison of two files, counting differences in the dependent - governor relation and computing the accuracy, precision and recall at exit.
- Java-based dependency tree viewer for simultaneous viewing of several dependency trees. This tools is used for "manual" evaluation of parsing results. It can also, e.g., highlight the differences between two parses.

Results and Achievements (an overview)

We believe that the main achievements of the summer Workshop'98 in the area of parsing are:

- 3 parsers working with a language different from English, all of which have improved upon a baseline set before the workshop
- confirmation of the hypothesis that the lexicalized CF parsing can be adapted for handling other languages, albeit with somewhat lower accuracy
- a set of observations regarding the "Parsing-by-translation" approach
- implemented the "Superparser" method for combining the ouput of several parsers, working at the moment by "bagging"
- data availability for further experiments by the team or by anybody else (the Treebank is now freely available)
- improved Czech tagger
- identified open problems regarding parsing Czech as a representative of highly inflective and free-word order languages.

The most important quantitative results obtained at the end of the workshop are summarized here:



The Results

- *Accuracy in %:*

Parser	01a	01b	02a	02b	09a	09b	05a	05b
Dev	71.9	79.3	51.5	55.3	54.7	68.2	77.0	+0.8
Eval	72.3	80.0	54.1	56.2	55.5	68.3	79.1	+0.8

- 01a: Collins, baseline
- 01b: Collins, final Workshop'98 version
- 02a: Zeman, baseline (tag dependencies only on dictionary tags)
- 02b: Zeman, lexical dependencies, machine disambiguated tags + wider beam during search
- 09a: Chelba/Schwartz: baseline
- 09b: Chelba/Schwartz: final Workshop'98 version (threshold 5 (unknown), MDt (p-sc))
- 05a: Brill/superparser, baseline(Collins' only for comparison; diff. version for dev-test and eval-test)
- 05b: Brill/superparser, forced best-only ('unbalanced': dev-test: precision 80.6, recall 76.0, eval: 81.8/79.1)
- [Assumed experiment: as 05b, using 01b for training on bags: should get up to (80.0 + 0.8 = ***80.8***)]

8/19/1998 WS 98 NLP Team 15

The Parsers

In this section we describe briefly each of the five parsing efforts. More detailed accounts can be found in the respective chapters of this report.

Lexicalized Context-Free Parser

(Chapter 1)

Recent work in statistical parsing of English has used lexicalized trees as a representation, and has exploited parameterizations that lead to probabilities which are directly associated with dependencies between pairs of words in the tree structure. Typically, a corpus such as the Penn treebank is used as training and test data: hand-coded rules are used to assign head-words to each constituent in the tree, and the dependency structures are then implicit in the tree.

In the Czech PDT corpus we have dependency annotations, but no tree structures. For parsing Czech we considered a strategy of converting dependency structures in training data to lexicalized trees, then running the parsing algorithms originally developed for English. Crucially, the mapping from dependencies to trees is one-to-many. The choice of tree structure is important in that it determines the parameterization of the model: that is, the independence assumptions that the parsing model makes. There are at least 4 degrees of freedom when deciding on the tree structures:

1) How "flat" should the trees be? The trees could be maximally flat (one phrasal level per head-word), binary branching, or anything between these two extremes.

2) What set of part of speech (POS) tags should be used?

3) What non-terminal labels should the internal nodes have?

4) How to deal with crossing dependencies?

As a baseline system we: 1) made the trees as flat as possible; 2) chose just the major categories (noun, verb etc.) as parts of speech; 3) derived non-terminal labels from the POS tag of the headword (for example, a phrase headed by a noun would be labeled NP); 4) effectively ignored the crossing dependencies problem.

During the workshop we refined this conversion process in several ways: modifying the tree structures for linguistically special cases such as coordination and relative clauses; experimenting with different POS tag-sets; and modifying the parsing model (for example, extending it to handle bigram dependencies at the phrasal level).

The baseline results on the final test set were 72.4% accuracy. The final system, with all refinements, recovered dependencies with 80.0% accuracy. Results on the development set showed that newspaper-style text (75% of the sentences in test data) were parsed at around 2% greater accuracy than this averaged, 80% result. These results therefore compare favorably to those on English Wall Street Journal text: that is, around 90% accuracy but with considerably more training data (890,000 words vs. 347,000 for Czech).

A "Direct" Dependency Parser

(Chapter 2)

This Workshop "sub"-project involved a parser for Czech based on direct statistical modeling of tag / word dependencies in Czech. In comparison to the Lexicalized CF parser, this does not use any grammar, and works directly with dependency trees instead of parse trees.

Several techniques were developed in preparation for this workshop, that had not been published before and that help the tag-based part of the parser. Although they were prepared before the workshop, the workshop brought a great deal to their implementing for the Prague Dependency Treebank, and to testing them thoroughly.

Then the second part of the workshop was devoted to lexicalizing the parser, i.e. to developing a new statistical model that deals with dependencies between words rather than between the morphological tags. It is true that this part did not help the parser as much as expected (and as reported for other parsers for English) but the outcomes are still challenging and the model developed here enables to continue with the research in future.

Let us very briefly look at the structure of the parser. Its main task can be characterized by the following expression:

$$\arg\max_T p(T|S) = \arg\max_T \left(p(S|T) \cdot p(T)\right)$$

It means that the parser wants to find the dependency tree T that maximizes p(T|S) where S is the sentence being parsed. In other words, we want to construct the tree that most likely is a dependency structure of the sentence S. Because in no way we are able to decide among all possible trees in the universe, we have to decompose the tree probability into edge probabilities. These can be estimated from the relative frequencies of dependencies in training data. Then, using the Viterbi search algorithm, we try to find the tree with the highest possible product of probabilities of its edges. Here we take a significant simplification that the dependency probabilities are statistically independent, i.e.

$$p(T) = \prod_{i=1}^{n} p(d_i)$$

This obviously is not true and weakens the parser so that we had to introduce various constraints, additional models and techniques that help us a little to work around this weakness. A list of them follows; a more detailed description will be given later in this report.

- Crossing dependencies (so-called non-projective constructions) are not allowed.
- A supervised reduction of morphological tag set was done. The number of different tags occurring in corpus decreased from about 1000 to about 400.
- A new model for valency was added. It says the parser how likely a node with a given tag has a particular number of child nodes.
- We take into account whether the words forming a dependency are adjacent in the sentence or not.
- We take into account whether the dependency goes to the right (the governing node precedes the dependent one in the sentence) or to the left.

The following table gives a brief summary of the results in terms of parsing accuracy. That is, each number is the percentage of dependencies generated by the parser that were correct. Unless stated otherwise, all the numbers characterize parsers trained on over 19000 sentences (approx. 300000 words) and tested on one of the two test data sets, the development test data, and the evaluation test data. The e-test data was used only at the very end of the workshop to cross-validate the results, so most stages have been tested with the d-test only. The training set and both the test sets contained texts from three different sources, from a daily newspaper, from a business weekly, and from a scientific magazine. It turned out to be much more difficult to parse the last one (mainly because of the sentence length) so it seems reasonable to give separate results for the scientific magazine (labeled "sci") and for the rest (labeled "norm").

The baseline parser includes all the techniques whose development started before the workshop so "baseline" may be read as "before lexicalization". A deeper description of the techniques and a more diversified summary of their contribution to the parsing accuracy will be given later in this report. The final results include the lexical part of the parser as well as some minor improvements that will be described later, too.

	D-test			E-test		
	norm	sci	all	norm	sci	all
Baseline	54	48	51	57	51	54
Final	57	52	55	58	53	56

The "Parsing by Translation" Approach

(Chapter 3)

We conducted a preliminary survey of the prospects of using bilingual corpus to generate a grammar for monolingual parsing. The purpose of the survey was primarily educational. We wanted to learn about the current state of the art in parsing as applied to large-scale corpora. We examined the Czech Readers Digest corpus, which consists of around 2 million words of aligned Czech and English sentences and has been preprocessed in Prague. Since we had parsers available for both languages, we parsed both sides of the corpus and surveyed the parse structures. Although we did not have time over the summer for a large-scale project to automatically generate one grammar from the other, we felt that there were enough systematic structural correspondences to warrant further work in this direction. The purpose of this report is to describe some details about this very valuable bilingual corpus and to comment on the tools we used for our survey.

We conducted a preliminary survey of how we might use a bilingual corpus to generate a grammar for monolingual parsing. In particular, we examined the Czech *Readers Digest* corpus, which consists of around 2 million words of aligned Czech and English sentences. These sentences were taken from the Czech *Readers Digest* articles paired with the original English *Readers Digest* articles from which they were translated. The essential idea is that correspondences in the bitext imply structure for the two monolingual halves of the text. In the ideal case, we would like to infer the structure itself, based on the knowledge that the aligned sentences are translations of each other, or perhaps to parse the English side and use those structures to infer the Czech parses. In these cases, we would have a way to build up a new treebank, based in part on an analysis of the English side of the bilingual corpus. Furthermore, since we do have the Czech treebank, we could compare any new results with it as a reference point. Our somewhat more modest goal was to see to what extent the English parses and Czech parses correspond, given the corpus, parsers, and other resources that are available at the workshop. Since very large Czech and English treebanks are now available, we are able to train parsers to parse both sides of the corpus.

Our original motivation for trying this experiment at the workshop is to provide a very different source of grammatical information for Eric's "Super Parser". Our expectation is that the Super Parser will make the best improvement over the individual parsers when the individual parsers behave very differently. Being able to infer the Czech parses from the English side of course would have met the criterion of being a very different source of grammatical information. Moreover, to the extent that such an exercise is possible, we would have a means of building trainable parsers for languages for which treebanks are not available, but for which bilingual texts are available.

Adaptation of the “Structured Language Model” for Parsing Czech

(Chapter 4)

The Structured Language Model (SLM) was introduced by Ciprian Chelba and Frederick Jelinek as a language model that uses a statistical parser in a left to right manner in order to exploit syntactic information for use in a language model as part of a speech recognition system. In traversing a sentence, the Structured Language Model produces a series of partial lexical parses whose exposed headwords are used instead of the previous two words in a trigram-like prediction process. It is conjectured that this language model could be easily modified to produce good complete parses for Czech. One of the attractive features of the parsing method used in this model is that it can be easily modified to handle “crossing” or non-projective dependencies, a feature of the Prage Dependency Treebank that our other parsers currently ignore. Chelba, who implementated his Structured Language Model in C++, was gracious enough to allow us access to his code in order to modify it for our purposes. During the workshop, with time constraints and the usual range of difficulties encountered, we had time only to modify this Structured Language Model to work properly with the Czech data, to experiment with unknown word statistics, and to use part of speech tags generated by a version of Jan Hajic’s exponential model statistical tagger. The results obtained during the workshop are encouraging as they were obtained with a version of the SLM which, though modified, is still not optimized for parsing.

The Structured Language Model consists of two main parts: a parser and a predictor. The model proceeds along a sentence in a left to right manner, constructing partial parses for the sentence prefix available at each word. These partial parses consist of binary branching lexical parse trees where each node is associated with a lexical item and a nonterminal or part of speech (POS) label. Each nonterminal label and lexical item pair that covers a partial parse is referred to as a headword. The predictor uses the last two exposed headwords over a sentence prefix to predict the next word in the sentence. With parameter reestimation Chelba and Jelinek report results that this technique does achieve significantly lower perplexities for the UPenn Treebank of Wall Street Journal text.

Partial parses are constructed in a binary manner by considering the last two headwords and their associated tag information. The parser can choose from three moves at any given point: ADJOIN-RIGHT, ADJOIN-LEFT, or NULL. An ADJOIN-RIGHT move creates a new nonterminal that covers the last two words, percolating the right word up the parse as the headword of the new phrase. ADJOIN-LEFT acts similarly, except that the left word is percolated up. After each adjoin operation, headwords are renumbered. The parser continues to build syntactic structure over a sentence prefix in this manner until the most probable move is the NULL move, which does not change the parse structure, but passes control to the predictor, which then predicts the next word and its POS tag from the previous two headwords. The model proceeds down each sentence in this manner until the end of sentence marker is reached. Because of the large (exponential) number of possible parse trees associated with any given sentence, this model uses a multiple stack search with pruning through the space of sentence parse tree hypotheses (see Chelba, Jelinek 1998). In this manner, hypotheses with equal numbers of parser operations and predictions are compared against each other.

Baseline results:

No unknown word statistics. No use of external (MDt = Machine Disambiguated tags) POS tags.

devel	54.7%
eval	55.5%

Use MDt tags for:

unknown word threshold:3	none	unk	all
devel	57.91%	67.42%	67.54%
eval	57.70%	67.16%	67.45%

unknown word threshlod:5	none	unk	all
devel	--	68.04%	68.18%
eval	57.29%	68.12%	68.32%

Various directions of future research are also discussed, such as handling crossing dependencies, parse closing strategies, optimization of the objective function, predictior probability decomposition, changes in data the annotation scheme, and optimization of the POS tagset.

The Superparser

(Chapter 5)

The goal of the superparser group was to explore the efficacy of combining the output of multiple parsers in hopes of generating a dependency structure of higher quality than that achieved by any single parser. Classifier combination has proven to be a powerful tool for improving the performance of machine learning algorithms and has recently been applied with success in natural language processing to a number of tasks, including part of speech tagging, word sense disambiguation and named entity recognition.

Our ultimate goal is to combine a diverse array of mature parsers. While we are currently developing a number of parsers for Czech (see elsewhere this report), at the moment one parser performs significantly better than the others, and appears to outperform the other parsers across all linguistic environments (reference table in superparsing section). Therefore, we will have to wait until the other parsers improve before we can expect gains through combination.

We instead focused primarily on diversification and combination of a single parsing algorithm, namely the Collins Parser (Chapter 1 of this report) ported to Czech. To generate a set of variants of the parser, we employed a technique known as bagging. Given a training set with N instances (sentences), we generate a single bag by randomly drawing instances from the training set N times with replacement. Once we have a training bag, we can then train the parser on that bag. We can generate as many bags as we wish, with each bag containing a slightly different version of the original training corpus, containing multiple copies of some sentences and no copies of others.

In the text of Chapter 5 we show the results achieved from generating multiple parsers via bagging and then combining the outputs of these parsers. We were consistently able to achieve an improvement in accuracy over the single parser trained from the original training set.

Future work

Several important problems have been identified during the work on the various parsers which warrant future research. The most challenging problem is the phenomenon of non-projectivity, or crossing dependency, which makes any parser based on a Context-Free Grammar theoretically insufficient. Also, the number of tags used for languages such as Czech is so high that we will have to work more on tag clustering or classification to obtain various clustering schemes suitable for various parsing techniques. Furthermore, the Czech tagger needs to be improved, especially for the number, gender and case categories, where it's error rate of 4-6% is still too high for parsing purposes.

Certain theoretical questions will have to be solved, too: for example, is the tree representation as chosen really the best for statistical parsing? It seems that at least for some of the parsers, some changes would help, while preserving the dependency structure which we still believe is best suited for further processing in natural language understanding tasks (i.e. *after* parsing is done).

Adn of course, the more data, the better: the annotation of the PDT continues and we will certainly re-run the experiments we did during the Workshop '98 when the PDT is finished and there is more than 1 mil. words for training.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. (#IIS-9732388), and was carried out at the 1998 Workshop on Language Engineering, Center for Language and Speech Processing, Johns Hopkins University. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or The Johns Hopkins University.

The resources (data and tools) used throughout the workshop are the results (in part or in whole, and on top of the funding provided by the workshop as described in the previous paragraph) of the following institutional grants (granted to the Institute of Formal and Applied Linguistcs, Faculty of Mathematics and Physics, Charles University, Prague, and lead by Prof. Eva Hajicova): Project No. VS96151 of the Ministry of Education of the Czech Republic, Grants No. 405/96/0198 and 405/96/K214 of the Grant Agency of the Czech Republic. The following individual grants have also contributed to the development of tools used at the workshop: Grant No. 39/94 of the Charles University, Prague, Czech Republic, and the Grant No. 195/1995 of the Research Support Scheme/HESP of the Soros' Open Society Fund.

The bilingual data used for the "Parsing by translation" task have been kindly prepared for use at the workshop by Martin Cmejrek and Jan Curin, now graduate students at the Institute of Formal and Applied Linguistics, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic. Special thanks to Reader's Digest VYBER (Prague, Czech Republic) for granting the licence for using their textual material.

The Prague Dependency Treebank data used at the workshop has been prepared and annotated by a team of people under the supervision of Jan Hajic (in alphabetical order): Alla Bemova, Eva Buranova, Jiri Karnik, Michal Kren, Petr Pajas, Jarmila Panevova, Jan Stepanek, and Zdenka Uresova. The morphological annotation of the data has been provided by a separate team, supervised by Barbora Hladka (in alphabetical order): Jiri Hana, Lenka Kebortova, Kristyna Kupkova, Pavel Kveton, Emil Jerabek, Jiri Mirovsky, Karel Skoupy and Stepan Verecky. Most of the technical work of the actual preparation of the data for the workshop has been done by (in alphabetical order): Alena Bohmova, Jan Hajic, Barbora Hladka, Kiril Ribarov and Dan Zeman, all of IFAL, Faculty of Mathematics and Physics, Charles University, Prague, before or during the workshop.

The Prague Dependency Treebank is based on raw texts provided by the Institute of Czech National Corpus, Faculty of Arts, Charles University, Prague, Czech Republic, which come from the following sources: Lidove Noviny Publishers, Mlada Fronta Publishers, Ringier CR, Academy of Science of the Czech Republic.

Last but not least, we are very grateful to the local organizers (above all to Jacob Laderman and Kimberly S. Petropoulos) who made all the technical and organizational arrangements necessary for a well-functioning and scientifically stimulating environment and thus made the Workshop'98 a very pleasant experience.

Chapter 1: Lexicalized PCFG: Parsing Czech

Michael Collins, Lance Ramshaw, Christoph Tillmann, Jan Hajič

Contents

Introduction

Recent work in statistical parsing of English has used lexicalized trees as a representation, and has exploited parameterizations that lead to probabilities directly associated with *dependencies* between pairs of words in the tree structure. Parsed corpora such as the Penn treebank have generally been sets of sentence/tree pairs: typically, hand-coded rules are used to assign head-words to each constituent in the tree, and the dependency structures are then implicit in the tree. In Czech we have dependency annotations, but no tree structures. For parsing Czech we considered a strategy of converting dependency structures in training data to lexicalized trees, then running the parsing algorithms originally developed for English. A few notes about this mapping between trees and dependencies:

- In general, the mapping from dependencies to tree structures is one-to-many: there are many possible trees with a given dependency structure.
- If there are any *crossing dependencies*, then there is no possible tree with that set of dependencies.

Input:
sentence with part of speech tags: I/N saw/V the/D man/N
(N = noun, V = verb, D = determiner)

dependency structure:

word		parent	word number		parent number
I	⇒	saw	1	⇒	2
saw	⇒	START	2	⇒	0
the	⇒	man	3	⇒	4
man	⇒	saw	4	⇒	2

Output: a lexicalized tree

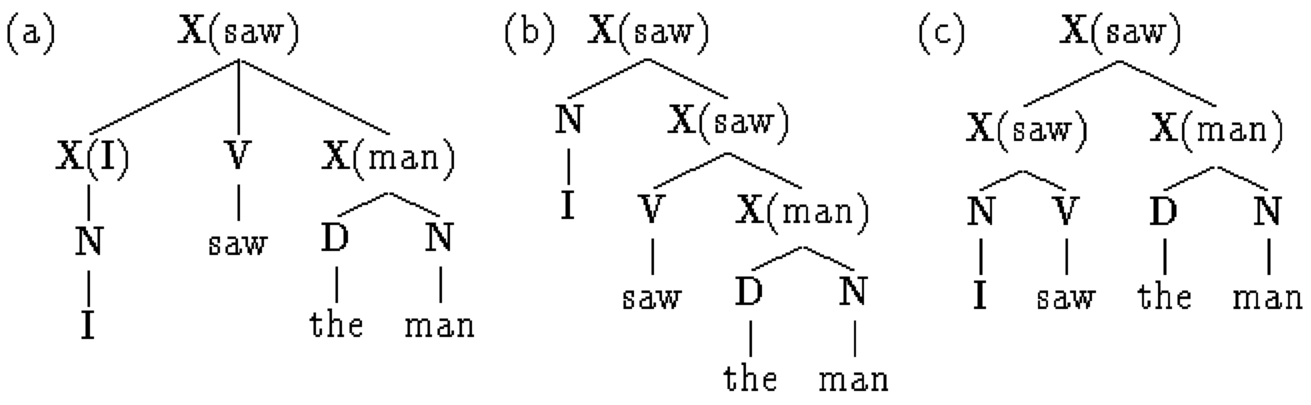


Figure 1: Converting dependency structures to lexicalized trees with equivalent dependencies. The trees (a), (b) and (c) all have the input dependency structure: (a) is the "flattest" possible tree; (b) and (c) are binary branching structures. Any labels for the non-terminals (marked *X*) would preserve the dependency structure.

Figure 1 shows an input dependency structure, and 3 lexicalized trees with this dependency structure. We explored different possibilities for the underlying tree structure for the dependency annotations. The choice of tree structure is crucial in that it determines the parameterization of the model: i.e., the independence assumptions that the parsing model makes. There are at least 4 degrees of freedom when deciding on the tree structures:

1. How "flat" should the trees be? The trees can be as flat as possible, as in figure 1(a), or binary branching as in trees (b) or (c), or somewhere between these two extremes.
2. What set of part of speech (POS) tags should be used?
3. What non-terminal labels should the internal nodes have? The non-terminals (*X* in trees (a), (b) and (c)) could take any value and still preserve the dependency structure.
4. How should we deal with crossing dependencies?

The Baseline Approach

To provide a baseline result we implemented what is probably the simplest possible conversion scheme, making the following assumptions:

1. The trees were as flat as possible, e.g., as in figure 1(a).
2. The part of speech tags were just the major category for each word (the first letter of the Czech POS set). For example, N = noun, V = verb, etc.
3. The non-terminal labels were "XP", where X is the first letter of the POS tag of the head-word for the constituent. See figure 2 for an example.
4. We ignored the problem of crossing dependencies: effectively any crossing dependencies in training data are dealt with by reordering the words in the string to give non-crossing structures (a natural consequence of the algorithm described later in this paper). The sentences in test data are (of course, for a fair test) left unchanged in order.

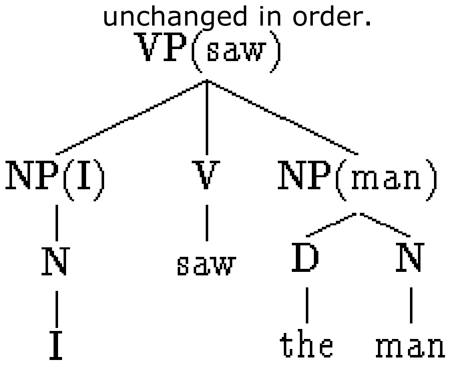


Figure 2: The baseline approach for non-terminal labels. Each label is XP, where X is the POS tag for the head-word of the constituent.

Section 7 gives pseudo-code for the algorithm. This baseline approach gave a result of 71.9% accuracy on the development test set. The next two sections describe a number of modifications to the tree structures that improved the model, and changes to the probability model itself.

Modifications to the baseline trees

Relative Clauses

In the Prague Dependency Treebank (PDT) the verb is taken to be the head of both sentences and relative clauses. Figure 3 illustrates how the baseline transformation method can lead to parsing errors with this style of annotation. The following algorithm was used to modify trees to distinguish relative clauses:

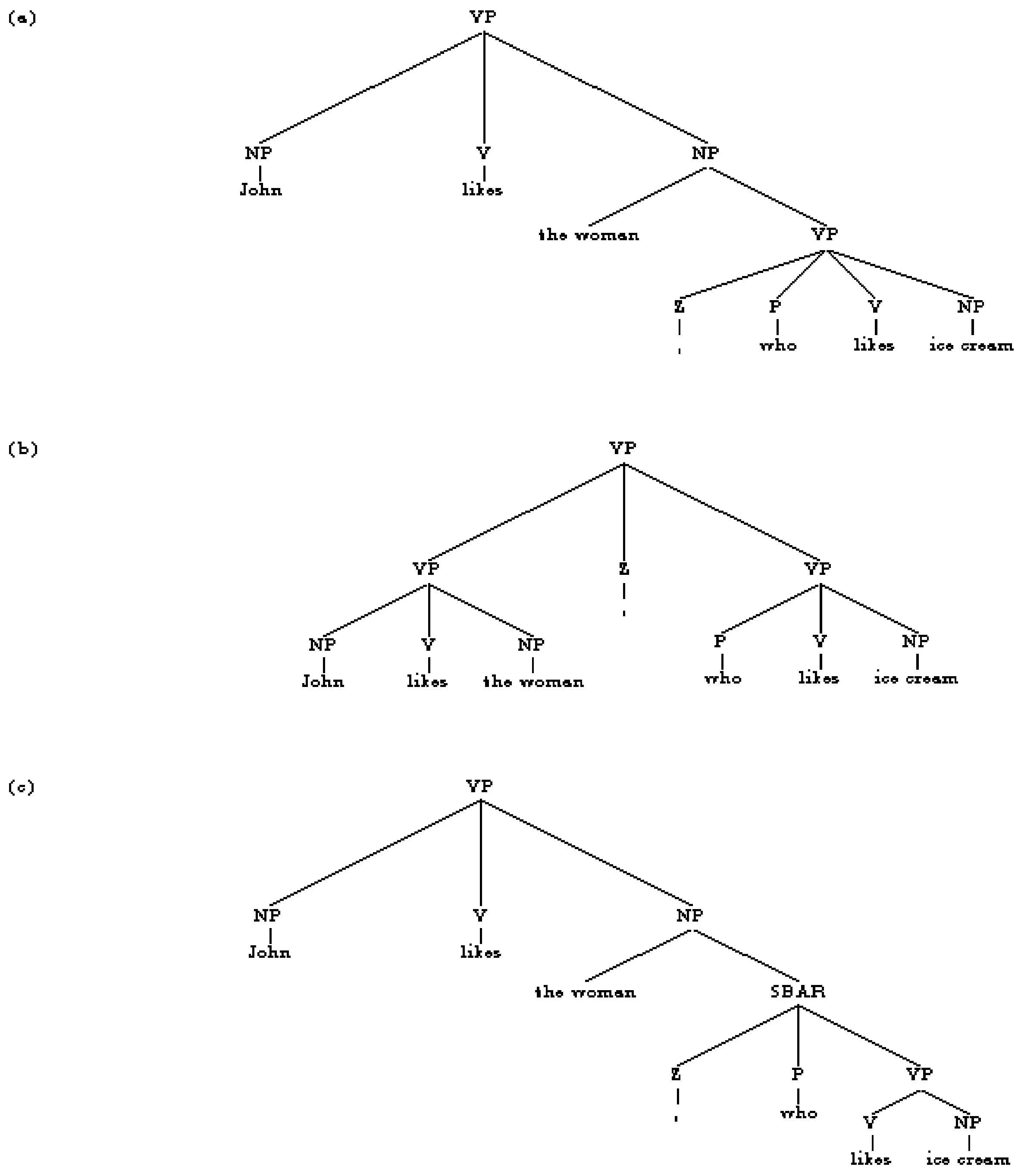
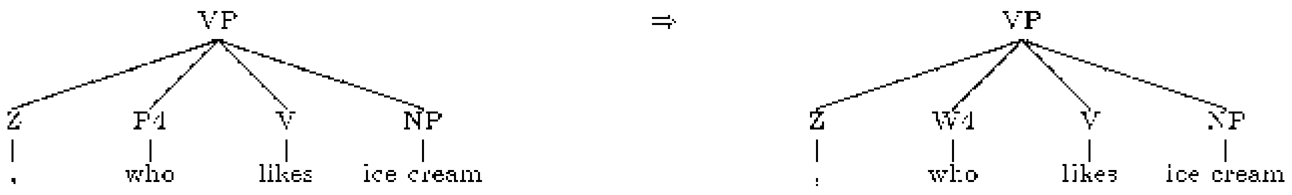


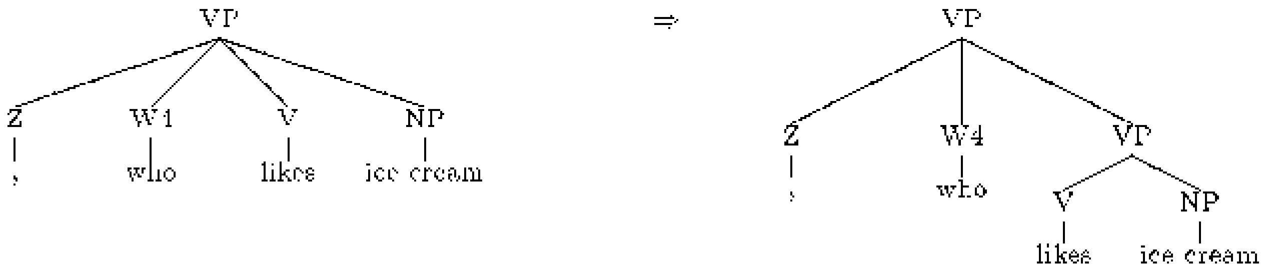
Figure 3: (a) The baseline approach does not distinguish main clauses from relative clauses: both have a verb as the head, so both are labeled VP. (b) A typical parsing error due to relative and main clauses not being distinguished. (note that two *main* clauses can be coordinated by a comma, as in *John likes the woman , the woman likes ice cream*). (c) The solution to the problem: a modification to relative clause structures in training data.

- Change the major POS category for all relative pronouns from **P** to **W** (**W** is a new category; relative pronouns have main POS **P** and sub-POS **1, 4, 9, E, J, K, Q, or Y**). For example,

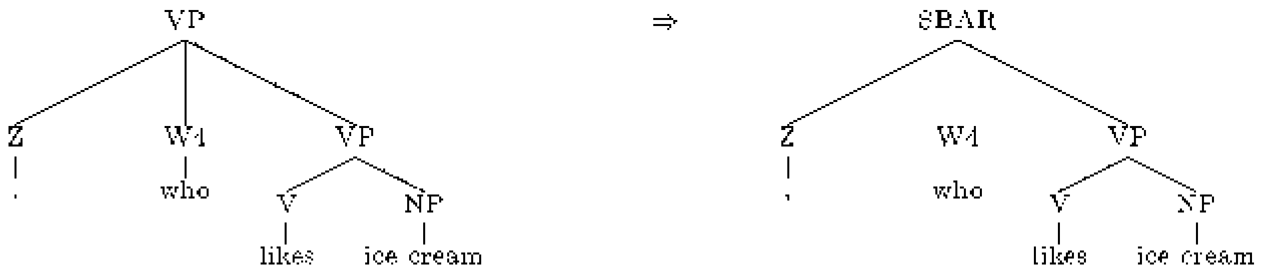


- For every relative pronoun dominated by a **VP**, with at least one sister to the right:

1. Add a new non-terminal labelled **VP**, spanning all children to the right of the relative pronoun. e.g.,



2. Change the label of the phrase dominating the relative pronoun to **SBAR**. e.g.,



Dealing with WH-PPs and WH-NPs

Relative pronouns are not the only signals of relative clauses: prepositional phrases containing relative pronouns (WH-PPs) or noun phrases containing relative pronouns (WH-NPs) can also be important --- see figure 4 for examples.

To account for these relative clauses we applied two stages of processing: 1) find all WHPPs and WHNPs in the training data, and change their non-terminal labels accordingly; 2) apply the relative clause transforms as before but treating WHPPs and WHNPs in the same way as relative pronouns.

Identification of WHPPs and WHNPs is done using the following recursive definitions:

- A WHPP is any PP which has a relative pronoun (POS tag with W as its first letter) as a child.
- A WHNP is any NP which has a relative pronoun (POS tag with W as its first letter) as a child.
- A WHNP is any NP which has a WHNP as a child.
- A WHPP is any PP which has a WHNP as a child.

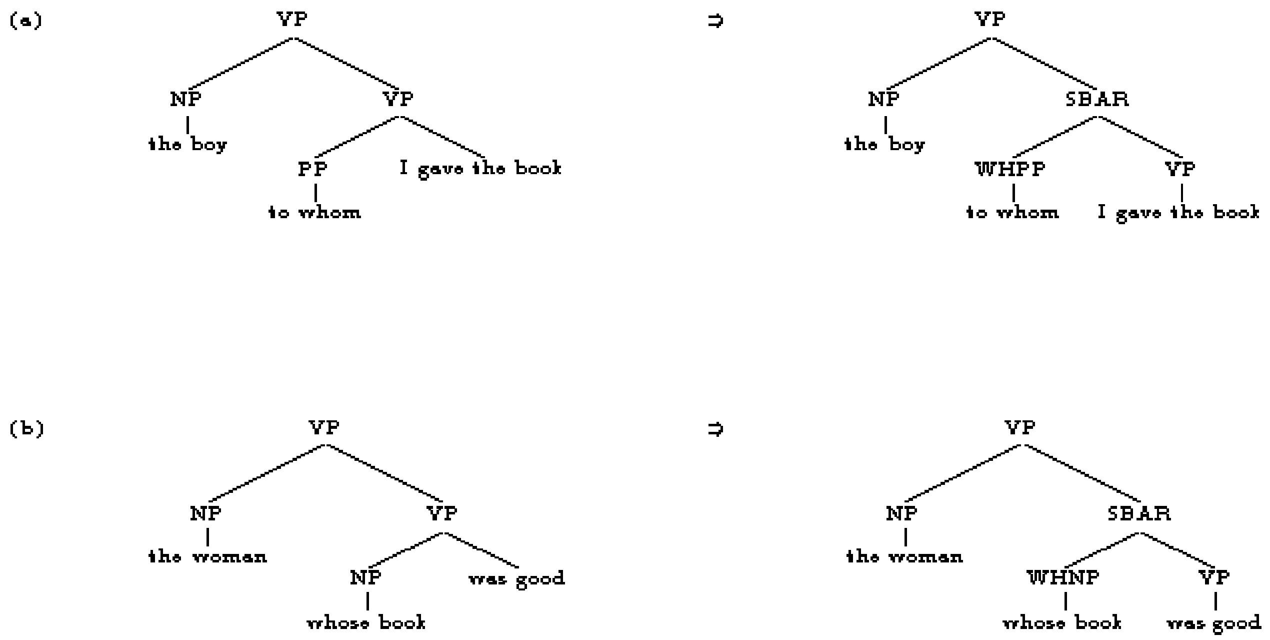
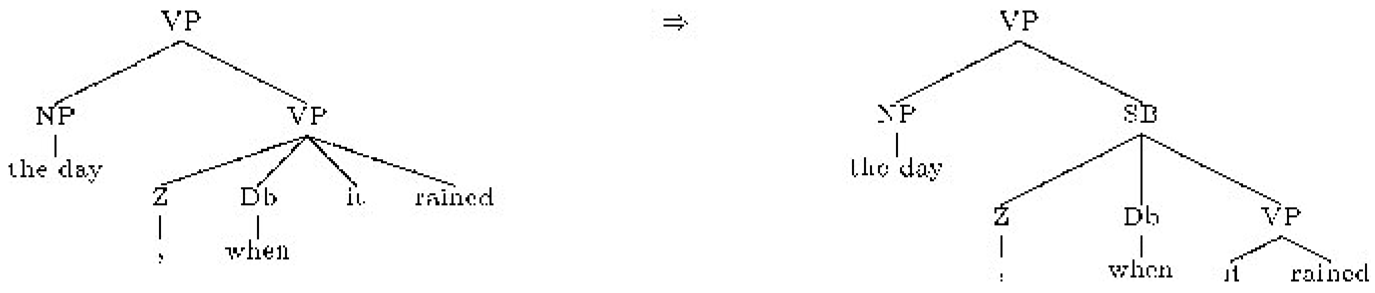


Figure 4: (a) An example of a PP, *to whom*, which has a relative pronoun as its argument. Its non-terminal label is changed to WHPP, and the phrase is changed to reflect it being a relative clause. (b) An example of an NP, *whose book*, which has a relative pronoun as a modifier. Its non-terminal is changed to WHNP, and the relative clause transforms are applied.

Dealing with WH-adverbials

An additional class of relative clause markers are wh-adverbials, for example *where* or *when* (as in *the country (SBAR where it rained)*, or *the day (SBAR when it poured)*. To treat these we identified cases where: 1) a phrase was a VP; 2) the first child of the phrase was a comma; 3) the second child was an adverb (tagged **Db**). A VP level was added covering all children following the **Db**, and the parent non-terminal was labeled **SB**. For example



Dealing with Coordination

To deal with cases of coordination two types of changes were carried out:

- For subtrees of coordination cases the internal labels were changed in order to improve the parameterization of the model (e.g. in Fig. 5, 7).
- New non-terminal labels were used within the tree structure to name new constituents - the orignal flat trees became more structured to improve the parameterization of the model. Examples are given in Fig. 8, 10.

Dealing with cases of coordination improved parsing accuracy by 2.6% as shown in table 3.

Handling of Coordination

When the head of the phrase is a conjunction, the original JP-non-terminal of the phrase is changed using the first letter of the non-terminal of the right-most child (see Fig. 5).

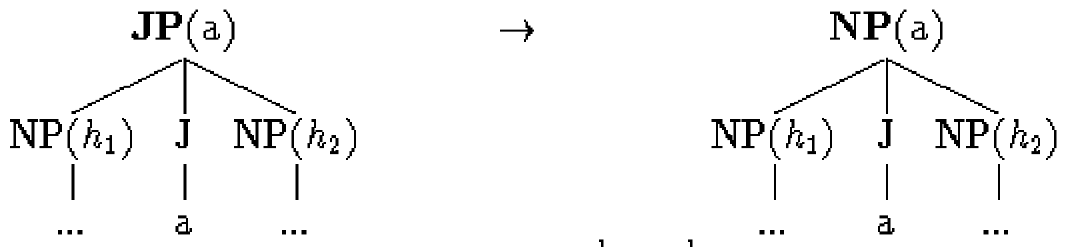


Figure 5: Example for handling conjunction: h_1 and h_2 are the headwords of the left and right NP.

Another change for coordination carried out was to create a new non-terminal based on the first letters of the left-most and the right-most children, e.g. in Fig. 6. After this change the parsing accuracy actually decreased. The most likely reason for that is the large increase in the number of non-terminals. The training data became too sparse for estimating the model parameters.

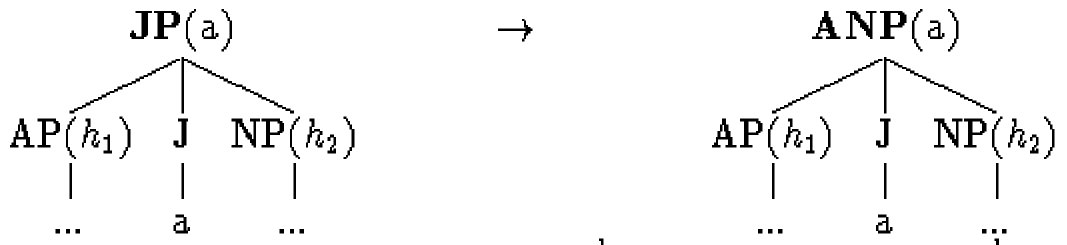


Figure 6: Example for handling conjunction: h_1 is the headword of the AP, h_2 is the headword of the NP.

When the head of the phrase is a comma, a colon, etc, the original ZP-non-terminal of the phrase is changed using the first letter of the non-terminal of the right-most child (see Fig. 7).

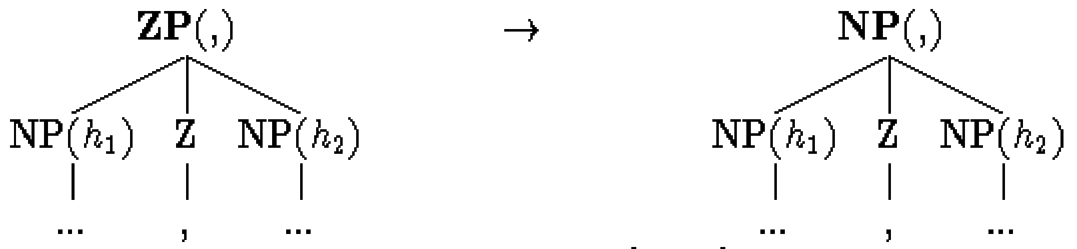


Figure 7: Example for handling punctuation: h_1 and h_2 are the headwords of the left and right NP.

Handling of Comma

A comma, which was the leftmost-child of a node, but not the head of the phrase, was treated in the following way: an additional non-terminal was added with the comma as the left child and the original phrase as the right child. This proved especially useful for subordinating sentences. The same change was carried out for a comma, which was the rightmost child of a given node, but accuracy was not improved.

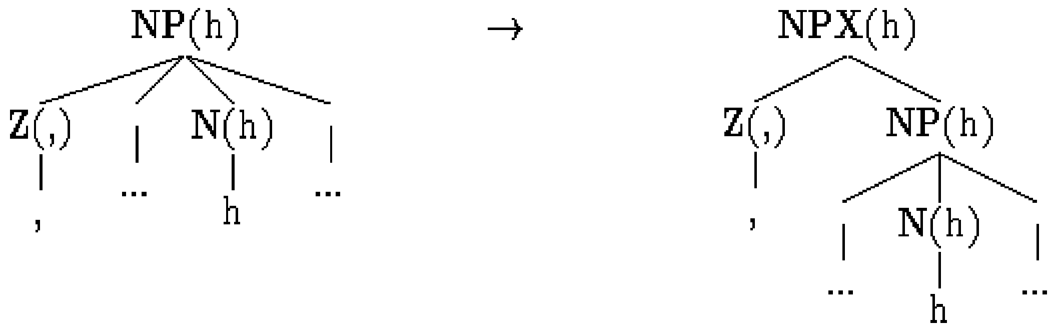


Figure 8: Example for handling a comma, which is the left-most child of a phrase: *h* is the headword of the NP. A new non-terminal 'NPX' is introduced.

Extended Handling of Coordination

In the case that within a coordination more than two elements were coordinated, additional structure was added to the tree. An additional non-terminal **XP,** was introduced, where *X* is a variable that can stand for any part of speech of the corresponding head-word, e.g. the tag **'N'** in Fig. 9. Including this change the model was made capable to learn that certain phrases tend to form chains of coordination. Surprisingly, this change did not improve the parsing accuracy.

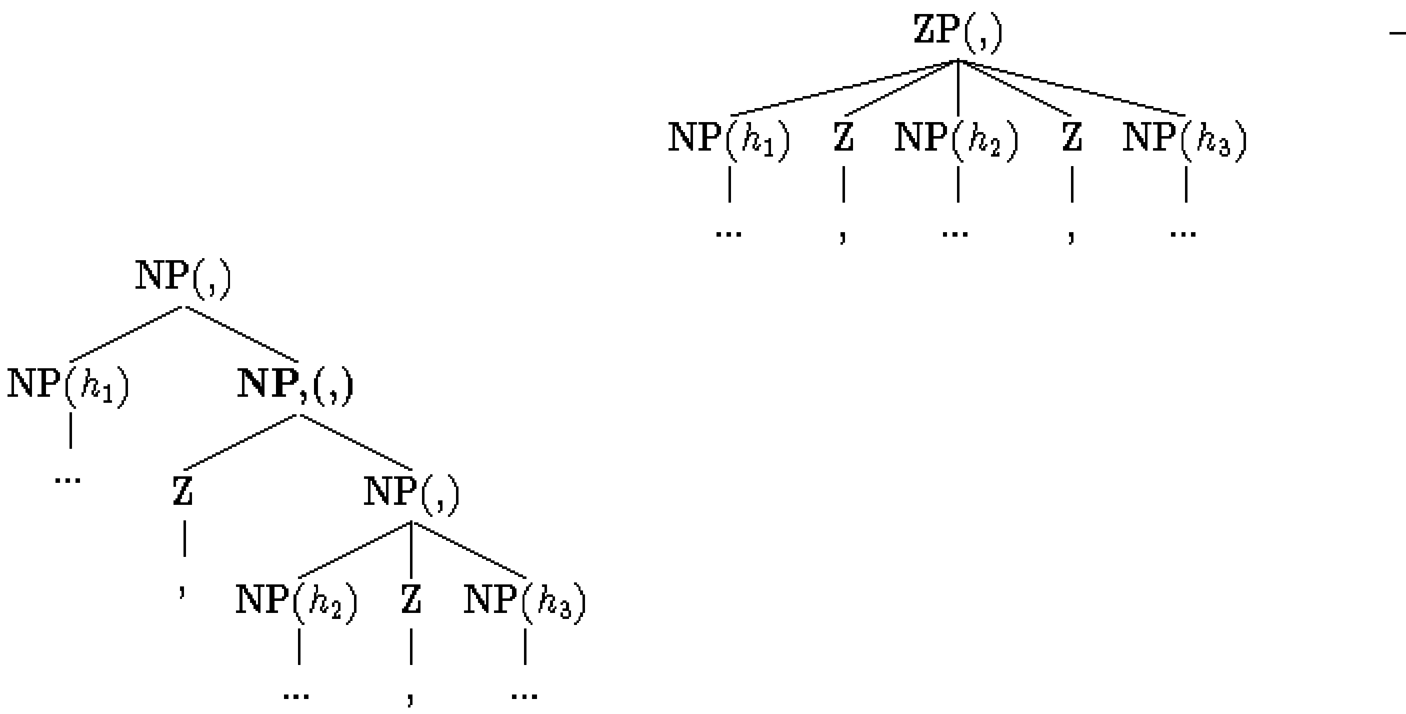


Figure 9: Example of handling several components in coordination: h_1, h_2 and h_3 are the headwords of the NPs. A new non-terminal 'NP,' is introduced.

Handling of Brackets and Quotation

Due to the zero-order assumption for the generation of right/left modifiers, left and right brackets are generated independently of each other. Since the generation process is a zero-markov-order dependency process, the model fails to learn that the pairs should be on the same level of the tree. We introduced a new label, e.g. NP-BRACKETS in Fig. 10, so that a pair of brackets is generated in parallel. For cases of quotation, where the quotes were on the same level of the tree an analogous new label NP-QUOTE was introduced. This change slightly improved accuracy by 0.2 %.

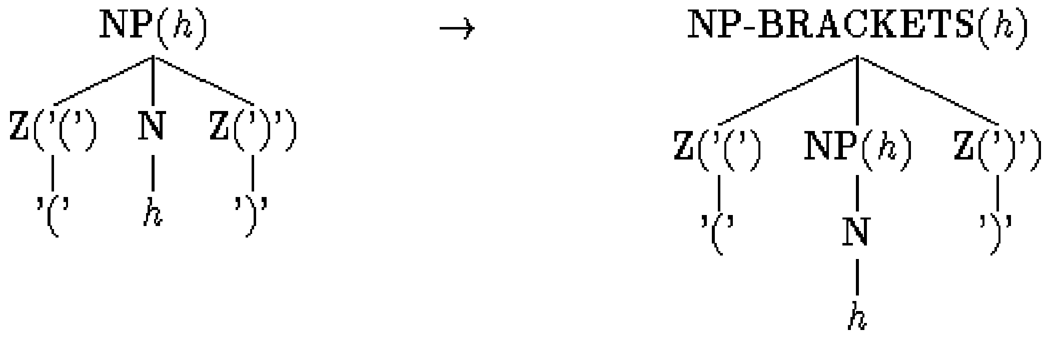


Figure 10: Example for handling brackets: *h* is the head of the NP.

Model Alterations

Preferences for dependencies that do not cross verbs

The models in (Collins 96, 97) had conditioning variables that allowed the model to learn a preference for dependencies which do not cross verbs. We weren't certain that this preference would also be useful in Czech, so we tried the parsing model with and without this condition on the development set (from the results in table 3, this condition improved accuracy by about 0.9% on the development set).

Punctuation for phrasal boundaries

It has been found that in parsing English it is useful to use punctuation (commas, colons, semicolons etc.) as an indication of phrasal boundaries (see section 2.7 of (Collins 96)). The basic idea is that if a constituent #tex2html_wrap_inline551# has two children *X* and *Y* separated by a punctuation mark, then *Y* is generally followed by a punctuation mark or the end of sentence marker. In the parsers in (Collins 96,97), this was used as a hard constraint. In the Czech parser we added a cost of -2.5 (log probability) in the following situation:

- If a constituent X takes another constituent Y as a pre-modifier, and: 1) Y is a comma/colon/semi-colon; 2) the last word of X is not a punctuation mark (tagged **Z**); 3) the word following X is not a punctuation mark; 4) the last word of X is not the last word of the sentence.
- If a constituent Y takes another constituent X as a post-modifier, and conditions 1, 2, 3 and 4 again apply.

Alternative Part-of-Speech Tagsets

Part of speech (POS) tags serve an important role in statistical parsing by providing the model with a level of generalization as to how classes of words tend to behave, what roles they play in sentences, and what other classes they tend to combine with. Statistical parsers of English typically make use of the roughly 50 POS tags used in the UPenn Treebank corpus, but the Czech PDT corpus used for this project provides a much richer set of POS tags, with over 3000 possible tags defined by the tagging system and over 1000 tags actually found in the corpus. Using that large a tagset with a training corpus of only 19,000 sentences would lead to serious sparse data problems. It's also clear that some of the distinctions being made by the tags are more important than others for parsing. We therefore explored different ways of extracting smaller but still maximally informative POS tagsets for use with the Collins parser.

Description of the Czech Tagset

The POS tags in the Czech PDT corpus (Hajic 98) are encoded in 13-character strings. Table 1 shows the role of each character. For example, the tag NNMP1-----A—would be used for a word that had "noun" as both its main and detailed part of speech, that was masculine, plural, nominative (case 1), and whose negativeness value was "affirmative".

1.	Main part of speech
2.	Detailed part of speech
3.	Gender
4.	Number
5.	Case
6.	Possessor's gender
7.	Possessor's number
8.	Person
9.	Tense
10.	Degree of comparisson
11.	Negativeness
12.	Voice
13.	Variant / register

Table 1: The 13-character encoding of the Czech POS tags.

Within the corpus, each word was annotated with all of the POS tags that would be possible given its spelling, using the output of a morphological analysis program, and with the single one of those tags that a statistical POS tagging program had predicted to be the correct tag (Hajic and Hladka 98). Table 2 shows a phrase from the corpus, with the alternative and machine-selected tag for each word. In the training portion of the corpus, the correct tag as judged by human annotators was also provided.

Form	Dictionary Tags	Machine Tags
poslanci	NNMP1-----A-- NNMP5-----A--	NNMP1-----A--

	NNMP7-----A-- NNMS3-----A-- NNMS6-----A--	
Parlamentu	NNIS2-----A-- NNIS3-----A-- NNIS6-----A-1	NNIS2-----A--
schválili	VpMP---XR-AA-	VpMP---XR-AA-

Table 2: Corpus POS tags for “the representatives of the Parliament approved”.

Selection of a More Informative Tagset

In the baseline approach, the first letter, or ;SPMquot;primary part of speech;SPMquot;, of the full POS strings was used as the tag. This resulted in a tagset with 13 possible values. A number of alternative, richer tagsets were explored, using various combinations of character positions from the tag string. Using the second letter, or “detailed part of speech”, resulted in a tagset of 60 tags. (The encoding for detailed POS values is a strict refinement of that for primary POS---that is, the possible detailed part of speech values for the different primary parts of speech do not overlap---so using the second letter alone is the same as using the first two letters together.) Combining the first letter with the fifth letter, which encodes case information, resulted in 48 possible tags. Each of these richer tagsets yielded some improvement in parsing performance when compared to the baseline “primary part of speech” tagset, but a combination of the two approaches did a bit better still 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, and provided about a 1.1% parsing improvement over the baseline on the development set. Even richer tagsets 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.

Explorations toward Clustered Tagsets

An entirely different approach, rather than searching by hand for effective tagsets, would be to use clustering to derive them automatically, We explored two different methods, bottom-up and top-down, for automatically deriving POS tag sets based on counts of governing and dependent tags extracted from the parse trees that the parser constructs from the training data. Neither tested approach resulted in any improvement in parsing performance compared to the hand-designed “two letter” tagset, but the implemetations of each were still only preliminary, and it might well be that a clustered tagset more adroitly derived could do better. The bottom-up approach to clustering begins with each tag in a class by itself. At each step, two classes are joined so as to maximize the average mutual information of the tag classes, as in the IBM work on “Class-Based *n*-gram Models of Natural Language” (Brown *et al.* 92), except that the mutual information here is calculated not over class bigrams in the text, but over pairs of governing tag and dependent tag, collected from the parse trees in the training set. Given the parsing model, the governing tag is the tag of the head of a consituent, and the dependent tag is that of the head of some subconsituent that is either a pre-modifier or post-modifier. The clustering process could be stopped at various cutoffs to test different sized tagsets. The top-down clustering began instead with a cluster tree of a single node, representing all tags in the same class. In each iteration, one leaf of the tree would be split into subclasses, choosing so as to maximize the total probability over all the parse trees in the training corpus of each governing tag generating its dependent tag time the probability of the dependent tag generating the word that actually occurred. (Thus our formulation of top-down clustering depended on the words in a way that our bottom-up clustering did not.) The algorithm relied on a fixed set of splitting rules, which were applied to each leaf at each step in order to select the best split. The splitting rules that were tested involved either a binary split based on whether a given character position in the tag matched a given value, or an n-ary split into as many children as there were values as a given character position. It is interesting to note that one early binary split found in the top-down clustering was based on position 12, which specified the voice of verbs, which had not been tested as a relevant variable in the hand-designed tagsets. As mentioned above, no clustered tagset was found that outperformed the initial two-letter hand-designed tagset, but this may have been due to problems in the inplementations. The comparative measures of parsing performance may also have been thrown off somewhat by other optimizations made to the parser that depended on the two-letter tags.

Dealing with Tag Ambiguity

One final issue regarding POS tags was how to deal with the ambiguity between possible tags, both in training and test. In the training data, there was a choice between using the output of the POS tagger or the human annotator’s judgment as to the correct tag. In test data, the correct answer was not available, but the POS tagger output could be used if desired. This turns out to matter only for unknown words, as the parser is designed to do its own tagging, for words that it has seen in training at least 5 times, ignoring any tag supplied with the input. For “unknown” words (seen less than 5 times), the parser can be set either to believe the tag supplied by the POS tagger or to allow equally any of the dictionary-derived possible tags for the word, effectively allowing the parse context to make the choice. (Note that the rich inflectional morphology of Czech leads to a higher rate of “unknown” word forms than would be true in English; in one test, 29.5% of the words in test data were “unknown”.) Our tests indicated that if unknown words are treated by believing the POS tagger’s suggestion, then scores are better if the parser is also trained on the POS tagger’s suggestions, rather than on the human annotator’s correct tags. Training on the correct tags results in 1% worse performance. Even though the POS tagger’s tags are less accurate, they are more like what the parser will be using in the test data, and that turns out to be the key point. On the other hand, if the parser allows all possible dictionary tags for unknown words in test material, then it pays to train on the actual correct tags. In initial tests, this combination of training on the correct tags and allowing all dictionary tags for unknown test words somewhat outperformed the alternative of using the POS tagger’s predictions both for training and for unknown test words. When tested with the final version of the parser on the full development set, those two strategies performed at the same level.

First-Order Dependencies

Our parser uses lexicalized-context-free rules of the following type:

$$\mathrm{P(h)} \Rightarrow \mathrm{L_1(l_1)} \ldots \mathrm{L_m(l_m) H(h) R_1(r_1)} \ldots \mathrm{R_n(r_n)}$$

H is the head-child of the phrase, which inherits the head-word *h* from its parent *P*. $L_1 \cdots L_n$ and $R_1 \cdots R_n$ are left and right modifiers of *H*. The generation of the RHS is decomposed into the three steps:

1. Generate the head constituent label *H*
2. Generate the modifiers to the right R_i of the head *H*
3. Generate the modifiers to the left L_i of the head *H*

For the english parser the following independence assumption was made: left and right modifiers were generated by seperate 0^{th} order Markov processes. The generation of a modifier does not depend on the already generated modifiers. The independence assumptions were changed to include bigram dependencies while generating the modifiers to the left and to the right. The generation of a left modifier L_i depends now on the immediately preceeding modifier L_{i-1} (the same is true for right modifiers). The head label *H* is generated as before.

1. Generate head label, with probability
$$\mathcal{P}_{\mathcal{H}}(H \mid P, h)$$
2. Generate left modifiers with probability
$$\prod_{i=1..n} \mathcal{P}_{\mathcal{L}}(L_i(l_i) \mid L_{i-1}, \mathrm{P,h,H})$$
3. Generate right modifiers with probability
$$\prod_{i=1..m} \mathcal{P}_{\mathcal{R}}(R_i(r_i) \mid R_{i-1}, \mathrm{P,h,H})$$

Due to the introduction of first-order dependencies several changes in the parser code became necessary:

- Parameter-Training: Get the counts to estimate the new bigram probabilities
- Dependency-Modelling: The back-off scheme to handle unseen events is changed
- Parsing-Algorithm: The dynamic programming algorithm for the chart parser is changed

The following lexicalized rule gives an example, for which we show its probability under the **zero-order** assumption and the **first-order** assumption:

S(bought) ⇒NP(yesterday) NP(IBM) VP(bought)

- Zero-order probability:
$$\begin{array}{ll}
\mathcal{P}_{\mathcal{H}}(\mathrm{VP} \mid \mathrm{S,bought}) & \times \\
\mathcal{P}_{\mathcal{L}}(\mathrm{NP(IBM)} \mid \mathrm{S,VP,bought}) & \times \\
\mathcal{P}_{\mathcal{L}}(\mathrm{NP(yesterday)} \mid \mathrm{S,VP,bought}) & \times \\
\mathcal{P}_{\mathcal{L}}(\mathrm{STOP} \mid \mathrm{S,VP,bought}) & \times \\
\mathcal{P}_{\mathcal{R}}(\mathrm{STOP} \mid \mathrm{S,VP,bought}) &
\end{array}$$

For the first-order assumption the modifier labels *NP* and *STOP* are generated using its immediately preceding modifiers, which are in both cases *NP*'s. *STOP* is a special modifier, which denotes the end of the generation of left/right modifiers.

- First-order probability:
$$\begin{array}{ll}
\mathcal{P}_{\mathcal{H}}(\mathrm{VP} \mid \mathrm{S,bought}) & \times \\
\mathcal{P}_{\mathcal{L}}(\mathrm{NP(IBM)} \mid \mathrm{S,VP,bought}) & \times \\
\mathcal{P}_{\mathcal{L}}(\mathrm{NP(yesterday)} \mid \mathrm{S,VP,}\mathit{NP}\mathrm{,\ bought}) & \times \\
\mathcal{P}_{\mathcal{L}}(\mathrm{STOP} \mid \mathrm{S,VP,}\mathit{NP}\mathrm{,\ bought}) & \times \\
\mathcal{P}_{\mathcal{R}}(\mathrm{STOP} \mid \mathrm{S,VP,bought}) &
\end{array}$$

The smoothing of dependency probabilities is illustrated using the following example. For illustration purposes we use linear interpolation for the smoothing. The actual implementation uses a Backing-Off scheme however. The example distribution is the next to the last term taken from the first-order probability.

$$\mathcal{P}_{\mathcal{L}1}(\text{STOP} \mid \text{S, VP, NP, bought}_V) =$$

$$\lambda_1 \tilde{\mathcal{P}}_{\mathcal{L}1}(\text{STOP} \mid \text{S, VP, NP, bought}_V) +$$

$$\lambda_2 \tilde{\mathcal{P}}_{\mathcal{L}1}(\text{STOP} \mid \text{S, VP, NP, V}) +$$

$$\lambda_3 \tilde{\mathcal{P}}_{\mathcal{L}1}(\text{STOP} \mid \text{S, VP, NP})$$

$$\lambda_4 \tilde{\mathcal{P}}_{\mathcal{L}1}(\text{STOP} \mid \text{S, VP})$$

The *STOP* non-terminal in the preceding rule example is predicted using the head-word 'bought' together with its part-of-speech *V*, the parent-label *S*, the head-label *VP* and the immediately preceding modifier *NP*. The more specific distributions are smoothed using the less specific ones.

Introducing the bigram-dependencies into the parsing model helped to improve parsing accuracy by about 0.9 % as shown in Table 3.

Results

The parser we used was model 1 as described in (Collins 97). We ran three versions over the final test set: the baseline version, the full model with all additions, and the full model with everything but the bigram model. The baseline system on the final test set achieved 72.3% accuracy. The final system achieved 80.0% accuracy: a 7.7% absolute improvement and a 27.8% relative improvement. The development set showed very similar results: a baseline accuracy of 71.9% and a final accuracy of 79.3%. Table 3 shows the relative improvement of each component of the model.

Type of change	Sub-type	Improvement
Tree Modifications	Coordination	+2.6%
	Relative Clauses	+1.5%
	Punctuation	-0.1% ??
	Enriched POS tags	+1.1%
Model Changes	Punctuation	+0.4%
	Verb Crossing	+0.9%

	Bigram	+0.9%
TOTAL	Absolute Change	+7.4%
	Error reduction	26%

Table 3: A breakdown of the results on the development set.

Table 4 shows the results on the development set by genre. It is interesting to see that the performance on newswire text is over 2% better than the averaged performance. The Science section of the development set is considerably harder to parse (presumably because of longer sentences and more open vocabulary).

Genre	Proportion (Sentences)	Proportion (Dependencies)	Accuracy	Difference
Newspaper	50%	44%	81.4%	+2.1%
Business	25%	19%	81.4%	+2.1%
Science	25%	38%	76.0%	-3.3%

Table 4: Breakdown of the results by genre. Note that although the Science section only contributes 25% of the *sentences* in test data, it contains much longer sentences than the other sections and therefore accounts for 38% of the *dependencies* in test data.

Pseudo-code for the baseline conversion algorithm

Input to the conversion algorithm

- *n* words $w_1 ... w_n$
- *n* POS tags $t_1 ... t_n$
- An *n* dimensional array *P*[1 ... *n*] where *P*[*i*] is the parent of w_i in the dependency structure. We assume word 0 is the start word.
- For each word an ordered list of its children (which can be derived from the array *P*). *Numchilds*[*i*] is the number of children for w_i. *Childs*[*i*][*j*] is the *j*'th child for w_i, where the children are in sequential order, i.e. *Childs*[*i*][1] ;*SPMlt*; *Childs*[*i*][2] ;*SPMlt*; *Childs*[*i*][3] ... NB. Childs[0], a list of children for the START word, is always defined also.

For example, for the dependency structure in figure 1:

- $w_1 ... w_4$ = {I, saw, the, man}
- $t_1 ... t_4$ = {N, V, D, N}
- *P*[1] ... *P*[4] = {2, 0, 4, 3}
- The childs/numchilds arrays:

i	*Numchilds*[*i*]	*Childs*[*i*]
0	1	{2}
1	0	{}
2	2	{1,4}
3	0	{}
4	1	{3}

Output from the conversion function

The output data structure is a tree. The central data type is a node, i.e. the node of a tree. This a recursive data type that specifies the node attributes, including recursive pointers to its children. The "tree" itself is simply a pointer to the top node in the tree. The **Node** data-type has the following attributes:

- **int type**. This is 0 if the node is internal (a non-terminal or POS tag), 1 if the node is a word.
- **int headword**. The head-word for the node --- if type==1 this is the word itself at this leaf.
- **int Numchilds**. The number of children for the node (by definition, this is 0 if the type==1).
- **Node **Childs**. An array of pointers to the children of the node, in left to right sequential order.
- **Node *Headchild** A pointer to the head-child of the phrase, which must also be in the Childs array.
- **char *label** The non-terminal label for the node.

The conversion function

The recursive function has a prototype

Node *create_node(int word)

This function takes the index *word*, which can take any value from 0 ... *n*, and returns a pointer to a node which is the top-most constituent with that word as a head (either a POS tag directly dominating the word, or a phrase with the word as its head). To create a tree, simply call tree = create_node(0); This will create a node whose head word is the 0'th (start) word, i.e. the entire tree. The pseudocode is as follows:

```
Node *create_node(int word)

{

//allocate memory for the POS tag node directly above word

create(Node tagnode);


//allocate memory for the word itself

create(Node wordnode);


//next create the word node
```

```
wordnode.type = 1;
wordnode.headword = word;

//next create the POS tag directly above the word
  tagnode.label     = t_word;
  tagnode.type      = 0;
tagnode.headword  = word;
tagnode.headchild = wordnode;
tagnode.numchilds = 1;
tagnode.childs[0] = wordnode;

//if the word has no dependent children in the dependency structure,
//then just return the POS tag as the node for this word
//(this will happen for words "I" and "the" in the figure)
if(numchilds[word] == 0)
return &tagnode;

//otherwise we'll create another node above the POS tag
create(Node phrasenode);

  phrasenode.label    = t_word + "P";
  phrasenode.type     = 0;
phrasenode.headword = word;
phrasenode.headchild = tagnode;

//Note the node has as 1 more child than the word has dependents,
//as there is an extra child for the head of the phrase, the POS
//tag for the word
phrasenode.numchilds = numchilds[word]+1;

//now recursively build the children
  //
//the one subtlety is that the head of the phrase, the tagnode
//that we have just built, has to be inserted at the right place
//in the childs sequence

n = 0;
//flag = 0 indicates the head of the phrase has not been inserted
//yet
flag = 0;
```

```
for(i=1;i <= numchilds[word];i++)
  {
if(flag == 0 &&
word < i)
    {
//insert the head node
phrasenode.childs[n] = tagnode;
n++;
flag = 1;
    }

//recursively create the sub-tree for the i'th dependent,
//and put it in the phrasenode.childs array
phrasenode.childs[n] = create_node(childs[word][i]);
  }

if(flag == 0)
  {
//insert the head node at the end of the phrase
phrasenode.childs[n] = tagnode;
  }

return &phrasenode;
}
```

References

1. [Brown et al. 1992]
   Peter Brown et al.: *Class-Based n-gram Models of Natural Language*, Computational Linguistics, Vol. 18, No. 4, pp. 467--479, 1993.
2. [Collins 1996]
   Michael Collins: *A New Statistical Parser Based on Bigram Lexical Dependencies*. In: Proceedings of the 34th Annual Meeting of the ACL, Santa Cruz 1996.
3. [Collins 1997]
   Michael Collins: *Three Generative, Lexicalised Models for Statistical Parsing*. In: Proceedings of the 35th Annual Meeting of the ACL, Madrid 1997.
4. [Hajic 1998]
   Jan Hajic: *Building a Syntactically Annotated Corpus: The Prague Dependency Treebank*. In: Issues of Valency and Meaning, pp. 106-132 Karolinum, Charles University Press, Prague, 1998.
5. [Hajic, Hladka 1998]
   Jan Hajic and Barbora Hladka: *Tagging Inflective Languages: Prediction of Morphological Categories for a Rich, Structured Tagset*, In: Proceedings of ACL/Coling'98, Montreal, Canada, Aug. 5-9, pp. 483-490, 1998.

Chapter 2: Workshop ‘98 Technical Report: Zeman’s Parser

Daniel Zeman

The subpart of this project which is internally called 9802 involved a parser for Czech based on direct statistical modeling of tag / word dependencies in Czech. In comparison to the 9801 parser, this does not use any grammar, and works directly with dependency trees instead of parse trees.

Several techniques were developed in preparation for this workshop, that had not been published before and that help the tag-based part of the parser. Although they were prepared before the workshop, the workshop brought a great deal to their implementing for the Prague Dependency Treebank, and to testing them thoroughly.

Then the second part of the workshop was devoted to lexicalizing the parser, i.e. to developing a new statistical model that deals with dependencies between words rather than between the morphological tags. It is true that this part did not help the parser as much as expected (and as reported for other parsers for English) but the outcomes are still challenging and the model developed here enables to continue with the research in future.

Let us very briefly look at the structure of the parser. Its main task can be characterized by the following expression:

$$\arg\max_{T} p(T|S) = \arg\max_{T} \left(p(S|T) \cdot p(T) \right)$$

It means that the parser wants to find the dependency tree T that maximizes p(T|S) where S is the sentence being parsed. In other words, we want to construct the tree that most likely is a dependency structure of the sentence S. Because in no way we are able to decide among all possible trees in the universe, we have to decompose the tree probability into edge probabilities. These can be estimated from the relative frequencies of dependencies in training data. Then, using the Viterbi search algorithm, we try to find the tree with the highest possible product of probabilities of its edges. Here we take a significant simplification that the dependency probabilities are statistically independent, i.e.

$$p(T) = \prod_{i=1}^{n} p(d_i)$$

This obviously is not true and weakens the parser so that we had to introduce various constraints, additional models and techniques that help us a little to work around this weakness. A list of them follows; a more detailed description will be given later in this report.

- Crossing dependencies (so-called non-projective constructions) are not allowed.
- A supervised reduction of morphological tag set was done. The number of different tags occurring in corpus decreased from about 1000 to about 400.
- A new model for valency was added. It says the parser how likely a node with a given tag has a particular number of child nodes.
- We take into account whether the words forming a dependency are adjacent in the sentence or not.
- We take into account whether the dependency goes to the right (the governing node precedes the dependent one in the sentence) or to the left.

The following table gives a brief summary of the results in terms of parsing accuracy. That is, each number is the percentage of dependencies generated by the parser that were correct. Unless stated otherwise, all the numbers characterize parsers trained on over 19000 sentences (approx. 300000 words) and tested on one of the two test data sets, the development test data, and the evaluation test data. The e-test data was used only at the very end of the workshop to cross-validate the results, so most stages have been tested with the d-test only. The training set and both the test sets contained texts from three different sources, from a daily newspaper, from a business weekly, and from a scientific magazine. It turned out to be much more difficult to parse the last one (mainly because of the sentence length) so it seems reasonable to give separate results for the scientific magazine (labeled ″sci″) and for the rest (labeled ″norm″).

The baseline parser includes all the techniques whose development started before the workshop so "baseline" may be read as ″before lexicalization″. A deeper description of the techniques and a more diversified summary of their contribution to the parsing accuracy will be given later in this report. The final results include the lexical part of the parser as well as some minor improvements that will be described later, too.

	D-test			E-test		
	norm	sci	all	norm	sci	all
Baseline	54	48	51	57	51	54
Final	57	52	55	58	53	56

The basic model

The backbone of the present parser is a statistical model that was first studied in [Zeman 1998 a]. (It is referred to as ″the unigram model″ there.) Let us now briefly summarize the fundaments of this model.

The main information the parser relies on (and thus the main thing to gather statistics about) is the dependencies between the words in a sentence. A **word** can be represented either by its **morphological tag** (containing the part-of-speech information as well as the inflection) or by some kind of **lexical information**. Or both.

During the training phase, the parser reads hand-annotated trees and saves the number of times a particular word was hanged on the other one. This way it estimates a probability for each dependency (tree edge) it sees in the data.

In the parsing phase, the parser tries to find the dependencies between tags so that the final tree has the highest possible probability. More formally, it searches for

$$\arg\max_{T} p(T|S) \quad (1)$$

where *S* is a sentence and *T* is its dependency tree. In the first approach we assumed that the only condition that bound *p(T)* with *S* was that the nodes of the tree had to be exactly the words of the sentence (except of the root — see the format of the Prague Dependency Treebank (PDT)). Otherwise, we dealt with an unconditional probability *p(T)*. Now there is a few new conditions using the word order information; they will be described later.

For simplicity the parser also assumes that the probability of the whole tree is a product of the probabilities of all dependencies in it. In other words, the probability of each particular dependency shall be statistically independent on any other edge in the tree:

$$p(T) = \prod_{i=1}^{n} p(d_i) \quad (2)$$

Obviously this is not true, so there is a significant loss of accuracy which we try to compensate with additional models and features. These will be described later.

1. Ambiguous morphological tags

In our approach the parser needs the input sentences to have morphological tags already assigned to each of the words. Unfortunately, word forms in Czech are highly ambiguous, so for many words there may be a lot of tags, each of them potentially describing the word in some context. Ideally we would need to apply a dictionary (or a morphological analyzer) which would assign a set of all possible tags to each of the words, and then a tagger that would say which tag from the set is true in this particular sentence. Nevertheless, the parser actually does not need a tagger. Instead of a unique tag, automatically assigned by a tagger, it *can* use the whole set from the dictionary.

The solution is as follows: When training, in each dependency we count all possible tag combinations. But they represent only one occurrence of an edge, so we add their numbers in register by *1/n* rather than by 1, where *n* is the number of tag combinations for the edge. When parsing, our estimation of the edge probability is the sum of the relative frequencies of all possible tag combinations for the given edge.

The tag ambiguity however increases the computational complexity. Most Czech words have ambiguous tags. Some words can have a relatively big set of them — for instance, there is a class of adjectives which all have an ambiguous form with 27 tags. Combined with lexical ambiguity (word form appurtenance to dictionary headwords), the number can yet increase, and a dependency between two such word forms is not very rare. So, to estimate one edge probability, the parser might need to access the table more than 700 times. Moreover, the tag ambiguity has influence on the accuracy of the results, as will be shown later.

2. Searching edges and building the tree

Obviously it is not trivial to select the optimal set of dependencies that maximizes the probability of the whole tree. Initially we used a greedy algorithm: in each step we added the edge that had the highest probability. Of course the set of allowed dependencies was constrained, the governing node had to be already added to the tree while the dependent node was not allowed to be there.

The greedy algorithm gives maximum probability only for non-oriented trees where $p(A,B) = p(B,A)$. For our case it is only an approximation, maybe too strong. It is impossible to find the real maximum (it needs exponential time) but we tried a Viterbi search. The set of all partially completed trees was truncated to N best of them. In each step we added M most promising (read: most probable) edges to each tree which gave $N \times M$ new trees; then the trees were sorted by their probabilities and the stack was truncated at the level N again. In most experiments (unless stated otherwise) we used $N=M=5$. Hopefully, in some cases this caused the most probable edge to be preserved for use in next rounds with even better gain.

The experiments showed that the initial model was so bad (31%, i.e. only some five points over a naive parser which creates chain-trees) that the Viterbi search did not improve anything. In fact almost all the generated trees had higher probability than the ”truth” from test data because the probability distribution modeled the reality badly. But as we added new features and additional models, the influence of the Viterbi search increased.

3. Projectivity and crossing dependencies

Let us now discuss the first constraint we imposed on the model. As mentioned before, the version 1 parser did not use the word order of the sentence. But the actual word order information is important for parsing even in such a ”free” word order language as Czech. For instance, in the generated parses it was quite often that a noun was connected to a preposition at the other end of the sentence, possibly with commas and verbs in-between, only because this particular dependency was relatively frequent in the training data. We cannot easily employ the absolute **distance** of the words to solve this problem because anytime there may be a phrase of an arbitrary length between the governing and the dependent node; however, there is another phenomenon that seems to be present with most of the ill-formed dependencies. It is called crossing dependencies or **non-projective constructions**.

Here is an example of a correct and a generated parse. The original sentence was *”Bohužel ale jednorázové, takže velkou èást spolknou danì.”* which can be translated as *”Unfortunately, [the donation was] given whole at a time so that a big part will be swallowed by taxes.”*. Note the crossing edges in the second tree. The floating point numbers in the second tree indicate the probabilities of the dependencies of every node on its parent.

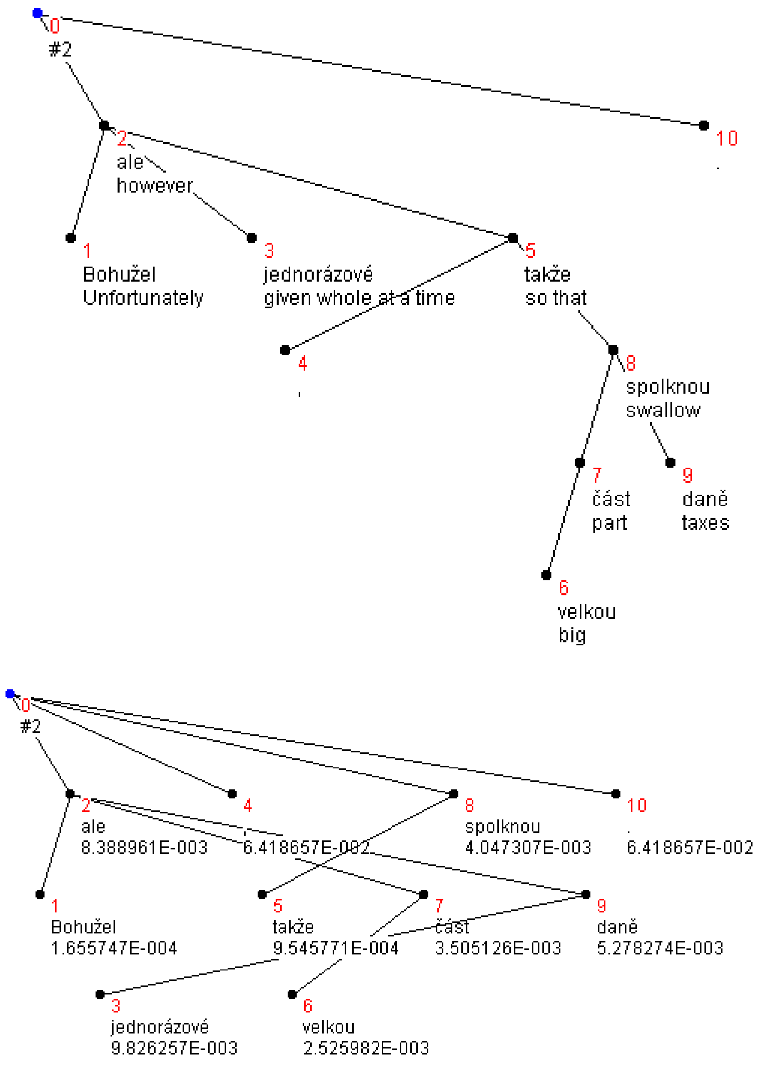
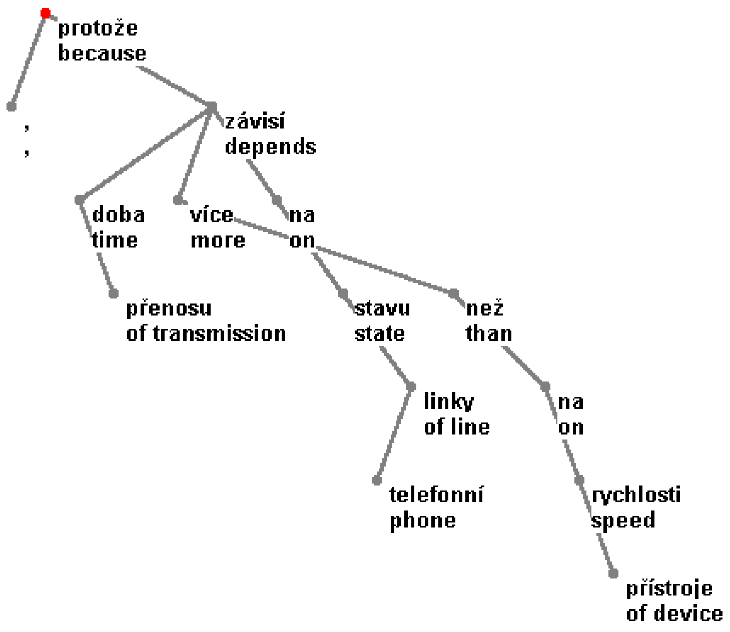


Figure 1: *A correct parse and a generated parse for the same sentence.*

Projectivity is a property that combines tree structure and the word order in sentence. A dependency A-B (where A is the governing node) is a **projective** construction if and only if all the words that are placed between A and B are included in the subtree of A. If you display the tree so that the x-coordinates of nodes correspond to the word order, each non-projective dependency will **cross** at least one perpendicular from another node. (It will not necessarily cross another dependency: in our example the nodes 3, 5, 7, and 9 are connected non-projectively.)



Let us mention that sometimes the non-projective constructions are *not* errors. The tree at the Figure 2 is an example of a *correct* parse with a crossing dependency. Such constructions are generally allowed in Czech but they are rare. Roughly 1.8% of all dependencies in the PDT are non-projective; the average length of a tree in the corpus is 16.3 words (dependencies). About 79.4% of all trees contain no crossing edge. The number of trees that contain one or no crossing dependency is 93.8%, and the number of trees with at most two such dependencies is 98.3%.

This investigation shows that it is a good idea to disable non-projectivities at all until the parser achieves an accuracy level beyond 90%. The experiments showed that using this constraint the parser improved its performance from 31 to 41%, resp. 42% (the first one using the greedy algorithm, the second one using the Viterbi search).

4. Proportional training

This chapter 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.

5. Reduced set of morphological tags

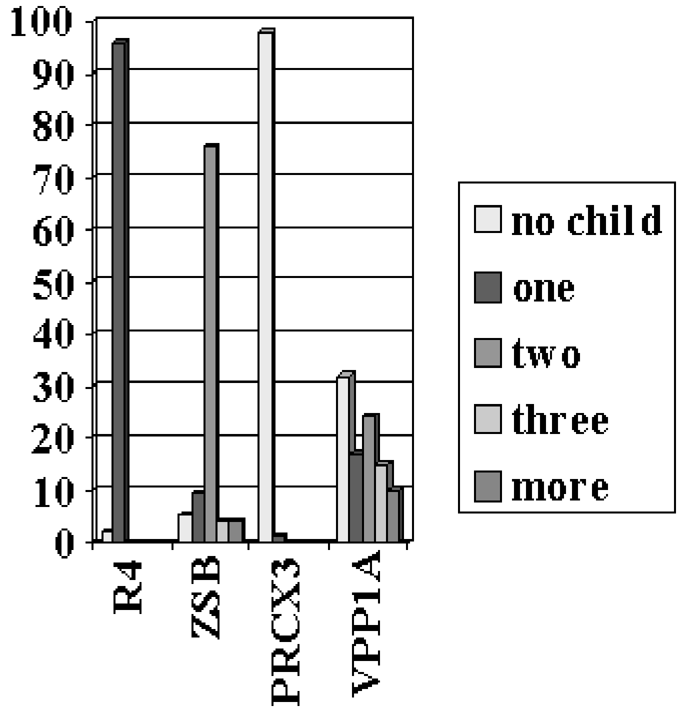
Some of the morphological information encoded in the tags has no influence on syntax. For instance, all adjectives, verbs and adverbs can have positive and negative forms, whereas the negative form is constructed completely regularly by simply inserting the prefix ”ne-” to the word beginning. This has no influence on the syntax because the negative forms behave syntactically exactly the same way that the positive ones do. So using such information in syntax training may damage the results. For instance, suppose we have seen a positive form of a word depending on a given other word n-times but we have not ever seen the negative form in this context. Then the edge with the negative form will get a probability of zero (or close to zero when smoothed) but in fact it should have the same probability as the one with positive form.

There are many other tags that we have merged together. Besides the negation, we also dropped degree of comparison of adjectives and adverbs, merged some classes of pronouns and numerals with nouns or adjectives, respectively, some classes of numerals match even adverbs. We also discarded information of some morphological and semantic subclasses (SUBPOS), and about the vocalization of prepositions. The VAR category (distinguishing some archaic, rarely used, and otherwise specific forms) had already been ignored. On the other hand, we split the tag for punctuation into several new tags in order to distinguish between commas, full-stops etc. Before the reduction there was about 1300 tags present in the training part of PDT, out of 3000 possible. After the reduction this number decreased to about 450 so the reduction rate is approximately 64%.

Note that this is a supervised reduction where a human has to say what shall be preserved and what shall not. There have been attempts to reduce the tag set using mutual information and a clustering algorithm. These two approaches have not yet been studied together and compared.

6. Valency

Some of the words tend to have many dependents while some others are almost all the times leaves, and the prepositions take mostly exactly one dependent — a head of a noun phrase. The parser described so far does not reflect this observations and, e.g., if there is one preposition in the sentence but several nouns, it often wants to hang all the nouns to the single preposition. To avoid this, we introduced an additional model which knows how often a given tag is a leave, how often it has one child node, how often two, and how often three or more. This is what we call ”valency”. The parser multiplies the edge probability estimations by the probability of the governing node having *n+1* children where *n* is the current number of children. With the ambiguous tags we use an average valency probability over all ambiguous tags for the governing node.



7. Direction and distance

Finally, we introduced two important features that reflect the mutual positions of the two dependency members in the sentence.

The first one is **direction**. The model assigns separate probabilities to a dependency where the depending node is (in the sentence) to the left of the governing node, and to the same dependency where it is to the right of the governing node. For instance, a preposition stands always to the left of its dependent. This feature (together with reduced tags and valency) pushes the accuracy to 49.6%; a more detailed summary of the results will be given later.

The second thing is distance, or better, **adjacency**. Now the parser only asks whether the two nodes are adjacent in the sentence or not. Separate probability estimates are kept for both choices. This feature (together with reduced tags and valency) pushes the accuracy to 48.5%.

Both direction and distance together bring the parser to 53% on the same small test data.

8. Summary of Version 2 results

Some of the features presented in the previous paragraphs immediately and significantly improve the results while some others work only when combined together. For this reason it was not always possible to specify the effect of a feature in terms of one number. Rather we present here a table of a number of feature combinations and the achieved accuracy levels. All experiments were run with the same small set of 199 sentences (trees) and 3036 words (dependencies).

Legend:

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

plain This model does not employ any of the following features. However, it disables the crossing dependencies, which is its only difference from the version 1 parser.

bt 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.

val Valency. A separate distribution of probabilities of number of child nodes for a given tag in the parent node.

prop Proportional training. In an edge that contained *n* possible tag combinations, each combination was counted with a weight corresponding to unconditional relative frequency of the tags participating on the combination. The former method counted each combination with a weight of *1/n* so there was a uniform distribution.

sou Adjacency (distance). For each particular tag dependency, both training and parsing phases distinguished the case, when both words were adjacent, from the other cases.

shr Dependency direction. For each particular tag dependency, both training and parsing phases distinguished the case, when the governing word lay left to the dependent word, from the case, when the governing word lay right to the dependent one.

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

The last table row shows the maximum performance of the ”version 2” parser that was mostly implemented during the preparation phase before the Workshop 98. During the first half of the workshop, it was adjusted to be compatible with the workshop data, and it was tested on larger sets of data. With the workshop d-test data set, it gave **51% accuracy**, which was the baseline for the workshop’s second part.

9. Different sources of morphological information

The version 1 parser worked with whole sets of ambiguous tags for each particular word, as assigned by the dictionary. The Workshop 98 version of the PDT allowed to use the disambiguated tags as well. Now there are two other sources of tags: a human annotation (available for training data only), and tags automatically assigned by a tagger (here a probabilistic one). We have mentioned that the ambiguous dictionary tags add computational complexity both in time and space. But we also confirmed the expectation that parsing with the dictionary tags is less accurate than with the disambiguated tags.

In these experiments we investigated different combinations of morphology sources for training and testing. Our training data set contained 13481 sentences. The tests were done on the workshop d-test data, i.e. 3697 sentences. The tagger was trained on approx. 200 000 words outside the treebank.

Training	Testing	Accuracy
13481 sentences	3697 sentences	
dictionary	dictionary	51.42
human	dictionary	52.59
human	tagger	53.44
dictionary	tagger	53.72
tagger	tagger	54.08

We were unable to test also the human-human combination because the test data are not annotated manually. However, we tried this with a small 124-sentence-set cut off the training data (of course, this portion had not been 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. Eventually 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 here the case).

The probability distributions for 13481 sentences

	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

Lexicalization

Finally, the second half of the workshop has been devoted to lexicalizing the parser. So far the parser made no use of lexical information at all: it relied completely on the morphological tags. But it is obvious that there are dependencies in which the lexical side could help a lot. Some notorious examples would be bindings of verbs, some preposition-noun pairs or idiomatic phrases at all. This expectation is supported by the results that are reported for other parsers with English: once they got lexicalized, their performance significantly improved (by about 10 percent points — see also [Charniak 1997]).

We wanted to build a new statistical model that would be similar to the present one in terms of collecting frequencies of particular word pairs but it would identify a word by its lexical contents rather than by the morphological tag. Then we would combine this new model with the old (morphological), and we would hope we'd get better results.

As we mentioned in the beginning, there are two different lexical attributes for each word. One of them is the **word form** as it appeared in the sentence. The other is the **lemma** or the dictionary headword that the form belongs to. Similarly to the morphological tags, the lemmas can be ambiguous, i.e. there can be two different lemmas that, together with some particular tags, can create the same word form. For instance, the word form ″je″ can be either the singular 3rd person present tense of the verb ″být″ (″to be″ — so the form means ″he is″) or the plural accusative of the pronoun ″on″ (″he″ — so the form means ″them″). However, the lemma ambiguity is not as high as the tag ambiguity. A probabilistic lemmatizer can do at 98% accuracy.

Now the question was whether to use the forms or the lemmas. The lemmas are a little ambiguous while the forms are not at all. On the other hand, the forms are much sparser than the lemmas. According to an electronic dictionary of Czech, the potential of the language would be about 700K of lemmas but about 20M of forms! However, in our training data only about 20K lemmas and 50K forms were seen. The ratio between possible and present will still increase to the power of two since we are working with word pairs rather than words. Another issue is that lemmas bear a different piece of information than tags while forms are in some sense subsets of tags. If we build our model with forms, we will be able to back off to tags whenever we fall against an unknown word pair. If we use lemmas, we shall treat the lexical and the morphological models as independent and combine them on a same-level basis.

We decided to use the lemmas. And similarly to the tags, for both training and parsing we used the lemmas automatically disambiguated by a lemmatizer.

10. Number of different lemmas and forms in the data

In the following table there are some figures supporting the arguments in the previous section. The columns represent different sorts of information: the lemmas assigned by a dictionary, by a lemmatizer or by a tagger, and the word forms. The rows give the numbers how many different entities of the respective category have been seen in the data *N* times or more; *N* is given in the first column. We can see that there are more word forms than the lemmas but they are sparser.

	Lemmas Dictionary	Lemmas Lemmatizer	Lemmas Human	Forms
³ 1000	24	25	24	19
100 ³	217	221	225	145

50 ³	528	530	528	364
25 ³	1207	1208	1207	879
5 ³	5165	5051	5071	5964
> 0	23474	21711	21647	58347

Once we decided to use the lemmas, we trained a model of automatically-assigned-lemma dependencies which had the following parameters (cf. parameters of the morphological models presented above).

The probability distribution for 13481 sentences

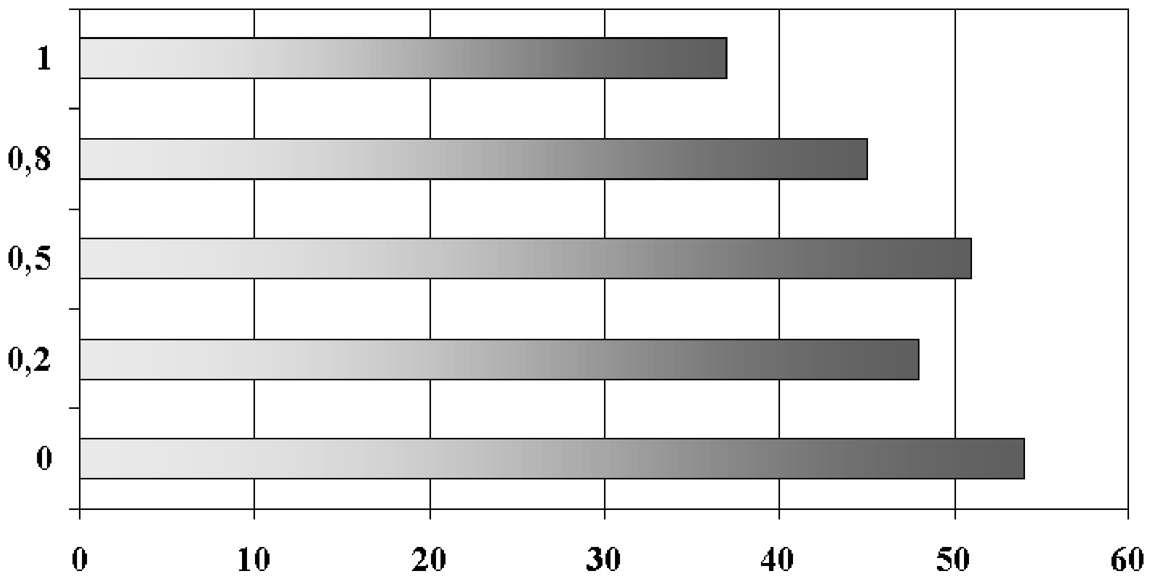
	Number of different dependencies	Maximum possible (uniform) entropy	Entropy	Perplexity
automatic lemmas	118827	16.86	15.22	38083

How to combine the lexical model with the morphological one

Keeping with the fact that lemmas and tags bear independent pieces of information, we did not want to use the old morphological model just as a back-off. Rather we tried to get a new probability distribution as a weighted interpolation of the two:

$$p(w_1, w_2) = \lambda \cdot p(l_1, l_2) + (1 - \lambda) \cdot p(t_1, t_2) \quad (3)$$

During the workshop we did not attempt to implement an automatic method of estimation of the l weight from held-out data. Instead, we experimented with several weights estimated by hand.



Unfortunately this use of the lexical information does not seem to help as expected. In the following table we can see the influence of the weight l . The number 1 means that the lexical model was used alone. It is not surprising that id did badly because the lemma dependency distribution is sparse. We had hoped that the lexical model would made different errors than the morphological one and that the combination of them would do much better. As we see from the table, the combination really did better but it was not able to exceed the accuracy of the morphological model alone. The result for the weight 0.5 forms a local maximum: this confirms that the both models really are good at different areas and that they are able to support each other. But the potential is eather not as great as expected or, more likely, it is not exploited entirely using this method. So the main task for the future work is to explore the methods of combination of two statistical model and to find a more suitable one.

In attempt to improve the accuracy at the workshop we introduced some minor changes. Firstly we stopped to trust lower counts. We considered unknown all dependencies that had been seen only five times or less. (The number five had been found experimentally: it gave the best results on the d-test data, together with four, while three and six worked worse.) We did not want to treat the unknown lexical dependencies the same way as impossible pairs. So in that cases the combined probability was taken as the morphological one with weight 1 (i.e. lexical weight zero) while for the ”impossible” (not seen at all) dependencies we still used the old scheme, i.e. half-half weights, thus the result was half the morphological probability. This pushed the accuracy to 54% but it still was not *better* than the version 2 results.

Then, finally, we changed the beam width (stack size) for the Viterbi search from 5 to 50. This of course lengthened the run time by ten but it improved the accuracy by one point. The 55% accuracy is the best we are able to achieve at the moment.

11. Results

The following table gives a summary of the results. Unlike the other experiments, the final runs were done with a parser trained on all the 19 126 sentences (327 597 words) and tested on both the development test data, and the evaluation test data. The e-test data was used first here at the end to cross-validate the results. The training set and both the test sets contained texts from three different sources, from a daily newspaper (*Mladá fronta Dnes* and *Lidové noviny*), from a business weekly (*Èeskoslovenský 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) so it seems reasonable to give separate results for the scientific magazine (labeled ”sci”) and for the rest (labeled ”norm”).

The baseline parser is actually the version 2 parser with all the features discussed in sections 1 – 9, except of the proportional training, which is now obsolete. Thus ”baseline” may be read as ”before lexicalization”. The only middle step is the shift from the dictionary-generated ambiguous data to the automatically disambiguated data. The final results include the lexical part of the parser as well as the minor improvements discussed after the lexicalization.

	D-test			E-test		
	norm	sci	all	norm	sci	all
Baseline	54	48	51	57	51	54
Disambiguated	56	51	54			
Final	57	52	55	58	53	56

The processing of the 3697 sentences of the d-test data by the version 2 parser takes on a 686 Intel equipped with Linux about 3 ½ hours. This number increases with the introduction of the lexical model and with making the Viterbi stack larger. Training the version 2 parser on 19000 sentences takes about 12 minutes. The time dramatically increases with the lexical model because the parser needs to build its lexicon during the training.

12. Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. (#IIS-9732388), and was carried out at the 1998 Workshop on Language Engineering, Center for Language and Speech Processing, Johns Hopkins University. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation or The Johns Hopkins University.

The author’s participation in the project, and the preparation of the annotated data, were also supported by the Grant Agency of the Czech Republic (Grant No. 405/96/K214), and by the Ministry of Education of the Czech Republic (project No. VS96151).

13. References

1. [Charniak 1997]
   Eugene Charniak: *Statistical Techniques for Natural Language Parsing*. In: AI Magazine, Volume 18, No. 4. American Association for Artificial Intelligence, 1997.
2. [Collins 1996]
   Michael Collins: *A New Statistical Parser Based on Bigram Lexical Dependencies*. In: Proceedings of the 34th Annual Meeting of the ACL, Santa Cruz 1996.
3. [Collins 1997]
   Michael Collins: *Three Generative, Lexicalised Models for Statistical Parsing*. In: Proceedings of the 35th Annual Meeting of the ACL, Madrid 1997.
4. [Zeman 1998 a]
   Daniel Zeman: *A Statistical Approach to Parsing of Czech*. In: Prague Bulletin of Mathematical Linguistics 69. Univerzita Karlova, Praha 1998.

zeman@ufal.mff.cuni.cz

Chapter 3: Working Notes on Exploring Parsing Resources from Bilingual Texts

Douglas Jones, Cynthia Kuo

Abstract

We conducted a preliminary survey of the prospects of using bilingual corpus to generate a grammar for monolingual parsing. The purpose of the survey was primarily educational. We wanted to learn about the current state of the art in parsing as applied to large-scale corpora. We examined the Czech Readers Digest corpus, which consists of around 2 million words of aligned Czech and English sentences and has been preprocessed in Prague. Since we had parsers available for both languages, we parsed both sides of the corpus and surveyed the parse structures. Although we did not have time over the summer for a large-scale project to automatically generate one grammar from the other, we felt that there were enough systematic structural correspondences to warrant further work in this direction. The purpose of this report is to describe some details about this very valuable bilingual corpus and to comment on the tools we used for our survey.

1. Motivation

We conducted a preliminary survey of how we might use a bilingual corpus to generate a grammar for monolingual parsing. In particular, we examined the Czech *Readers Digest* corpus, which consists of around 2 million words of aligned Czech and English sentences. These sentences were taken from the Czech *Readers Digest* articles paired with the original English *Readers Digest* articles from which they were translated. The essential idea is that correspondences in the bitext imply structure for the two monolingual halves of the text. In the ideal case, we would like to infer the structure itself, based on the knowledge that the aligned sentences are translations of each other, or perhaps to parse the English side and use those structures to infer the Czech parses. In these cases, we would have a way to build up a new treebank, based in part on an analysis of the English side of the bilingual corpus. Furthermore, since we do have the Czech treebank, we could compare any new results with it as a reference point. Our somewhat more modest goal was to see to what extent the English parses and Czech parses correspond, given the corpus, parsers, and other resources that are available at the workshop. Since very large Czech and English treebanks are now available, we are able to train parsers to parse both sides of the corpus.

Our original motivation for trying this experiment at the workshop is to provide a very different source of grammatical information for Eric's "Super Parser". Our expectation is that the Super Parser will make the best improvement over the individual parsers when the individual parsers behave very differently. Being able to infer the Czech parses from the English side of course would have met the criterion of being a very different source of grammatical information. Moreover, to the extent that such an exercise is possible, we would have a means of building trainable parsers for languages for which treebanks are not available, but for which bilingual texts are available.

2. The Czech *Readers Digest* Corpus

We started with the larger set of sentences: about 23 thousand sentences that were perfectly aligned and processed in Prague. We begin looking for isomorphic or nearly isomorphic parses. We also looked at small set of around 50 sentences by hand.

Some sample sentences are shown in Figure 1.

to je zvláštní , pomyslel si .	that 's strange , he thought .
sáhl na sklo a zjistil , že chvění může sice	feeling the glass , he discovered he could dampen the vibration but could n't make it

zmírnit , ale že ho nezastaví úplně .	stop .
‟ vylepšuju ho , ‟ odpověděl les .	‟ i’m making it better , ‟ les responded

Figure 1. Sample of Aligned Sentences

The sentences in the aligned corpus were perfectly matched. Consequently, some of the text from the original *Readers Digest* text that did not match was left out. Also, the formatting had been removed and the text was in all lower case. This presented a problem because the parsers and taggers were trained on mixed-case text. However, since we also had access to the full original text, we were able to reconstruct the formatting information from the original. We felt this was easier than re-aligning the original text to get the full formatting.

As you can see in (2), the sentences corresponded quite closely in length. The mean sentence length for the English sentences is 17. Five words whereas the median Czech sentences is a little over 16 words. The English sentences were a little bit longer. The reason for that appeared to be that various English function words (such as particles, preposition, determiners, and so on) corresponded to morphological inflections in Czech.

	English length	Czech length	Absolute Difference
Mean	17.5	16.1	7.6%
Median	16	15	6.7%
Stdev	8.8	8.2	6.5%

Figure 2. Comparison of Sentence Lengths

3. Exploration of Data Space

We divided the data space into two main categories: correspondences, those that can be transformed and those that cannot. Of those that can be transformed, some small number will actually be isomorphic structures. The remainder is the transformable structures. Of those that cannot be transformed in a straightforward way, some will be because of the inherent freedom of translation. Others will be because of parse errors or alternative analyses that are incompatible.

Recall that the Prague Dependency Treebank encodes dependency relations. The Penn Treebank, on the other hand, encodes constituent structures. We were able to produce both dependency structures and constituent structures for both the English and the Czech sides of our corpora, using the conversions developed at the workshop. For our initial survey, we looked at the dependency structures.

We will now step through examples of each type of data. Figure 3 illustrates what we mean by an isomorphic parse. What is required is that the topology of the parse be the same (of course) and that the nodes at each point correspond. The second half is not entirely trivial since the parts of speech do not correspond perfectly in Czech and in English. Figure 3 shows an example of an isomorphic parse for both sides of the corpus.

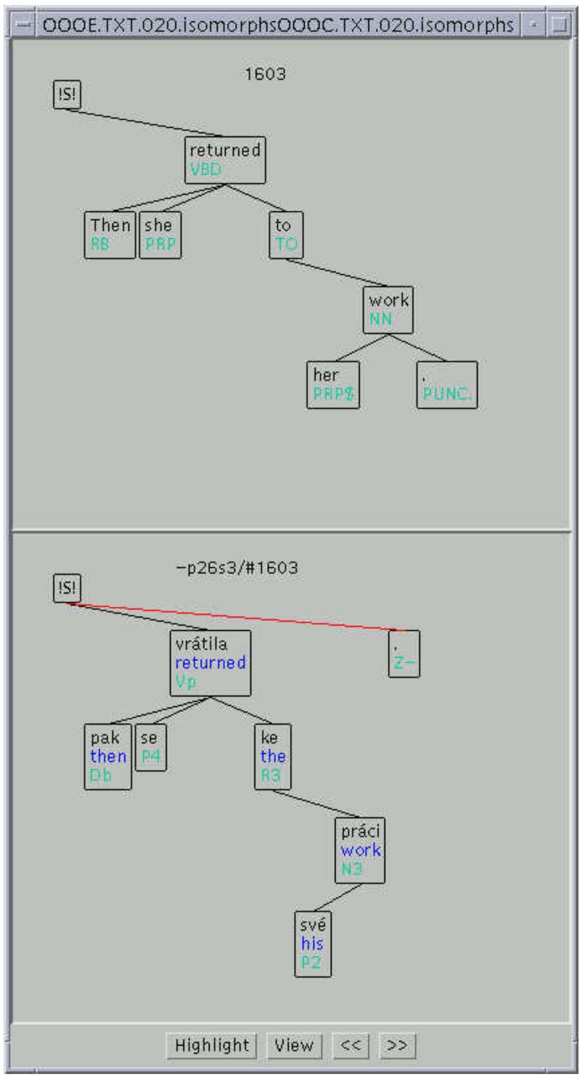


Figure 3. Isomorphic Parse

Transformable Parses

Naturally, isomorphic parses were very rare. We focussed most of our attention on parses that appeared easily transformable. In Figure 4, the only problem is that in the Czech parse, the punctuation is attached high in the tree, whereas in the English parse, it is attached low. This does not reflect a linguistic difference, but rather, a difference in encoding schemes in the two treebanks. Regardless, this is the kind of parse that would be easy to transform.

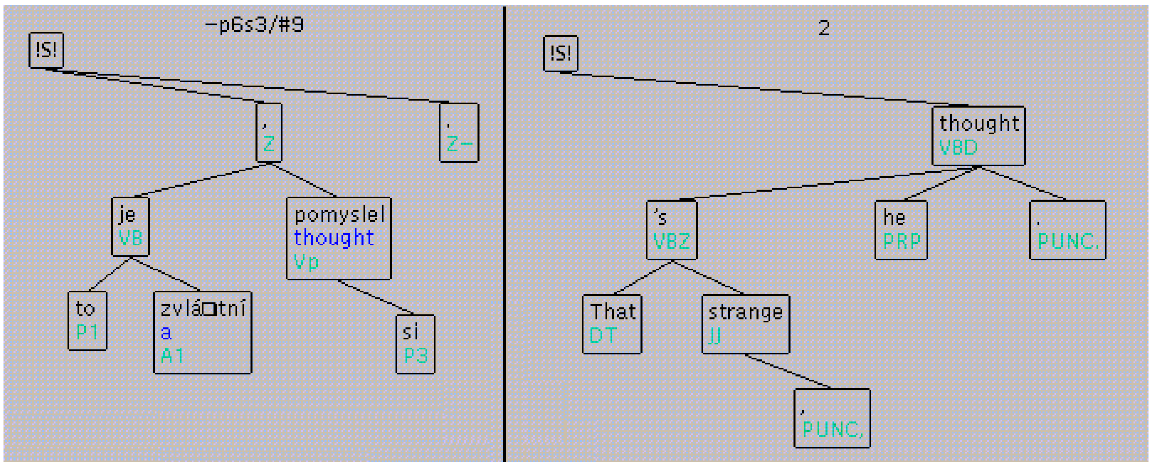


Figure 4 contains a sketch of some of the types of transformations that could be applied to create isomorphic correspondences.

Transformation	Applicable language	Reason
Paraphrasing / changing word order	Czech & English	Linguistic differences. In particular, Czech is a free word order language
Eliminating determiners	English	Czech does not use determiners, such as "the" or "a." Some determiners are, however, translated as pronouns in Czech.
Combining modal verbs and infinitives with main verb	English	Because of Czech's rich morphology, modal verbs and the infinitive "to" do not exist in Czech; the main verb is inflected.
Eliminating punctuation	Czech & English	Design differences between Penn Treebank and Prague Dependency Treebank. The Czech translations also contain added punctuation.
Skip prepositions	English	Where English uses a preposition, Czech may use a case marker on the noun (object of the preposition).
Dropping subjects	English	Czech sometimes drops the subject of a sentence, when the inflection on the main verb makes the subject clear.

Figure 4. Sketch of Transformation Types

Free Translations

Because of the inherent freedom in translation, some of the sentences correspond only very loosely. Figure N shows an example of such a "free" translation.

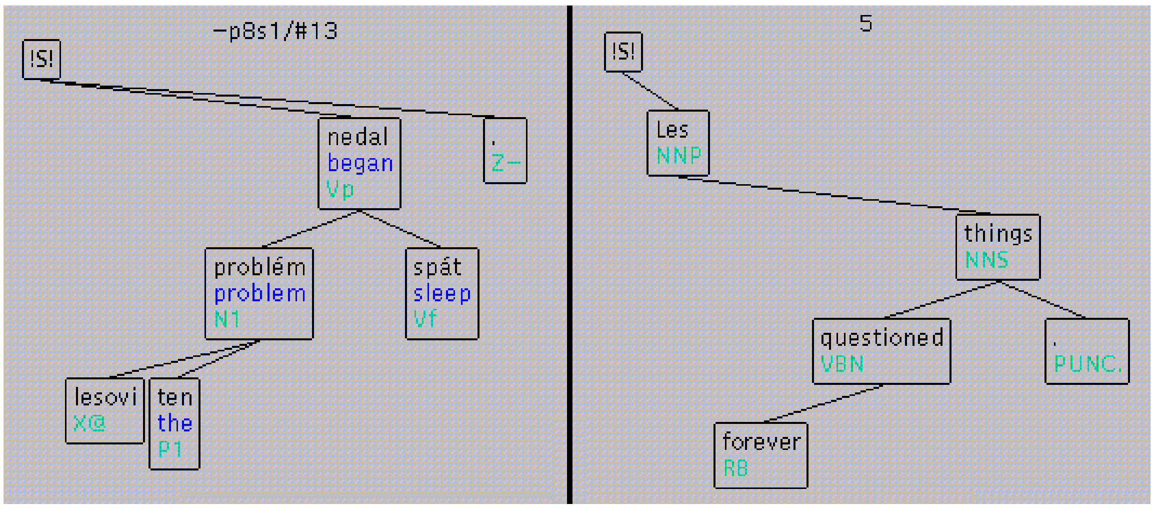


Figure 5. Free Translations not easily Transformable

Incompatibilities from Alternative Analyses

In some cases, the two treebanks simply encode relations differently. In the Penn Treebank, the category of a coordinate structure matches the elements coordinated. For example, the category of *Noun and Noun* is itself *Noun*. When the constituents are converted to dependency structures, the result is that the head of the phrase is one of the coordinates. In the Prague Dependency Treebank, on the other hand, the head of the coordinated structure is the coordinator. So the head of *Noun and Noun* is *and*, not *Noun*. Naturally, these structures do not match. We do expect them to be transformable.

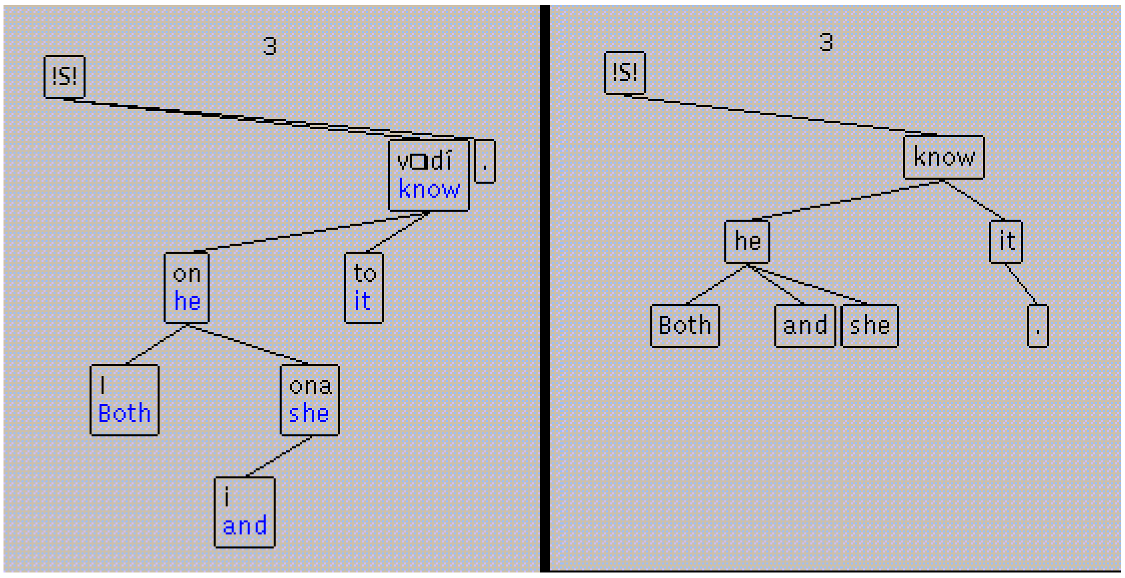


Figure 6. Alternative Analyses

Parse Errors

Since neither parser is perfect, we expect some of the mismatches in structure to be because of parse errors. The accuracy rate for Michael Collins's parser was 88% on the English data and 79% on the Czech data. The English sentence in Figure 7 is parsed incorrectly, possibly because of a tokenization error. *I* is tagged as a noun, when it should be a pronoun. Also, *it* and *better* should modify the verb *make*, not *respond*

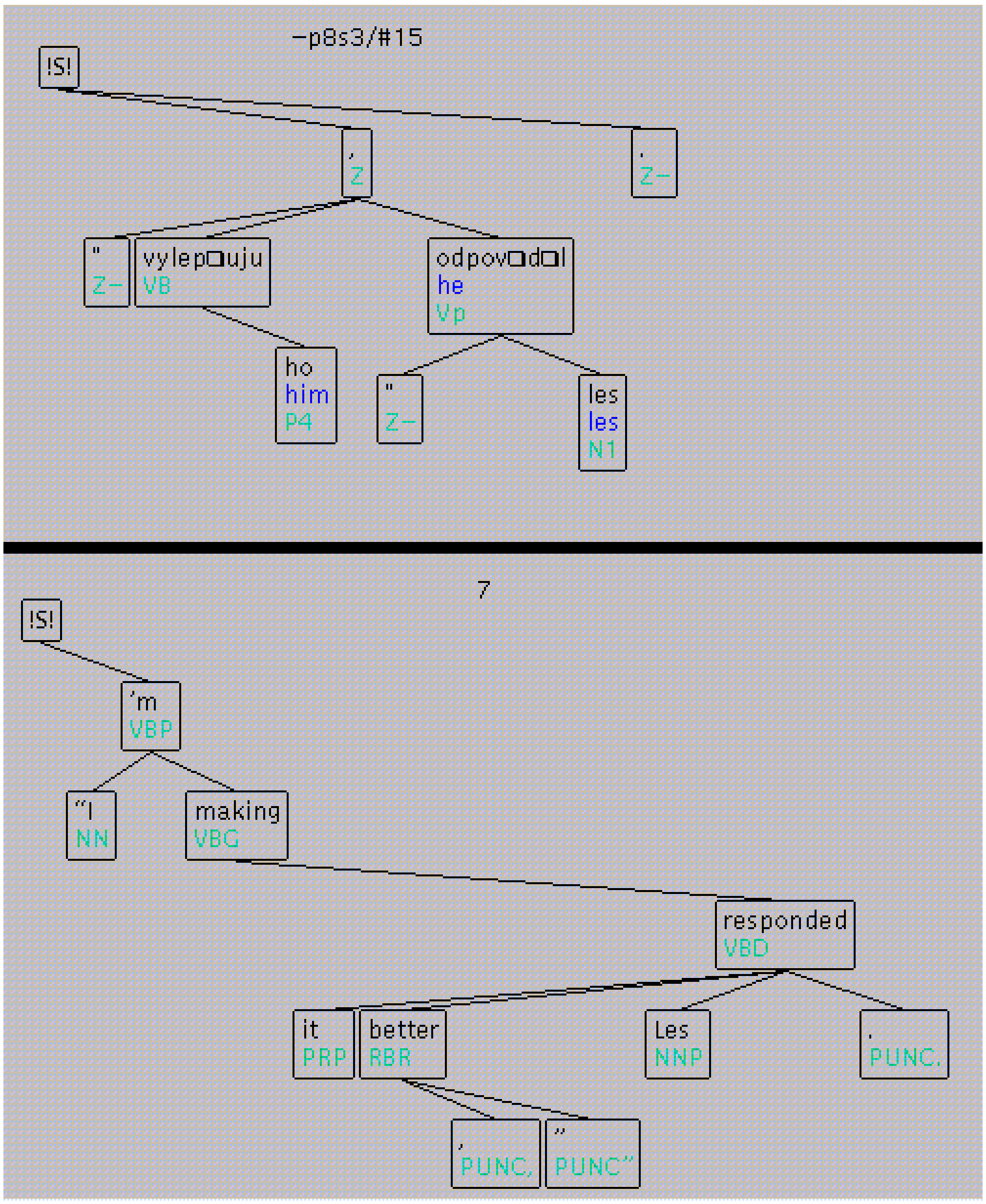


Figure 7. Parse Errors

4. Tools and Data Format

Preparing to conduct these experiments required some data formatting work to be done. Although substantial preprocessing for the text was already done in Prague, we still needed to get the English side into the proper format for parsing. Tagging and tokenizing presented some obstacles, since the aligned sentences had been pre-tokenized, but in a way that was not completely compatible with what the Collins parser needed for input.

To process a portion of the corpus, we usually created a Unix Makefile (see bin\Makefile) to apply each process to the original. The Makefile would take care of the following processing:

Alignment

For the alignment, we used the two files E000.TXT and C000.TXT, which contained one pair of perfectly aligned sentences on each line. These sentences were in lower case.

Tokenization

The problem with the lower case aligned sentences was that the Collins parser had been trained on the Penn Treebank, which was in upper and lower case. Our solution was to restore the original formatting of the Readers Digest articles. We felt this was easier than re-aligning the text. After restoring the formatting, we re-tokenized and tagged the text with standard tools.

- Czech: bin\restore-cz-sgml.prl
- English: bin\restore-en-sentences.prl, bin\tokenize-en

Tagging

For the English side, we used Eric Brill's tagger to tag the text, and adjusted the output to match the nonterminal symbols from the Collins parser. For the Czech side, we used the tagger from the workshop.

- Czech: bin\tag-cz
- English: bin\tag-en

Parsing

We were then able to parse the English Readers Digest text with Model 2 of the Collins parser. We were able to produce both constituent trees and dependency trees for the parses.

- Czech: bin\parse-cz, bin\just-parse.prl
- English: bin\parse-en, bin\just-parse.prl

Viewing the Parses

We then used Oren Schwartz's viewer to look at the parses. (The viewer is available at http://www.clsp.jhu.edu/ws98/projects/nlp/doc/9807/PV.html)

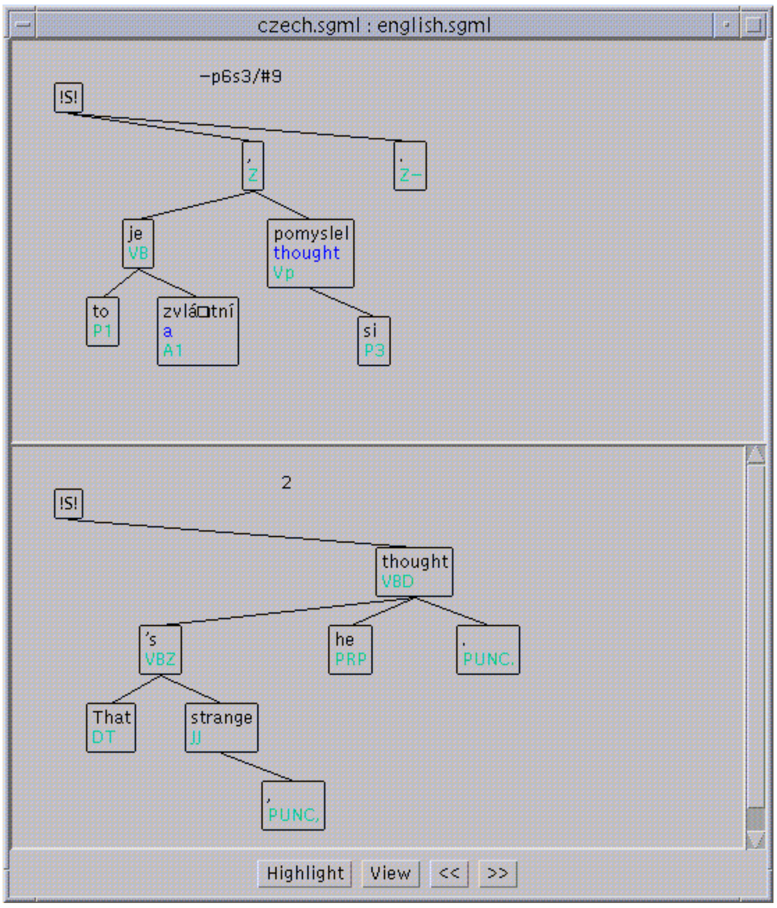


Figure 8. Oren Schwartz's Tree Viewer

We also used Radu Florian's tool for looking at the constituent structures: (shows partially matching trees).

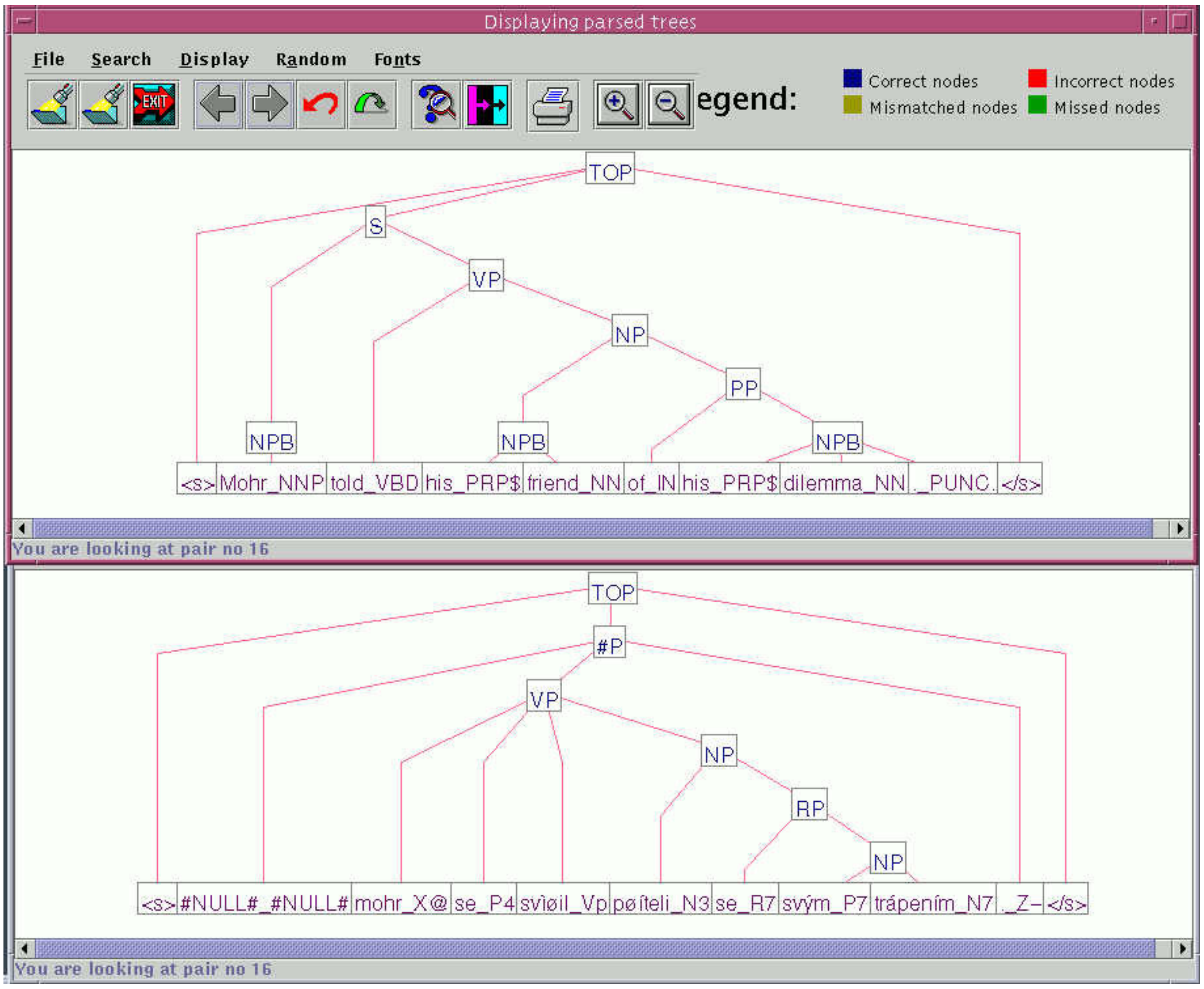

Displaying parsed trees
File Search Display Random Fonts
Legend:
🟦 Correct nodes
🟥 Incorrect nodes
🟨 Mismatched nodes
🟩 Missed nodes
TOP
S
VP
NP
PP
NPB
NPB
NPB
<s> Mohr_NNP told_VBD his_PRP$ friend_NN of_IN his_PRP$ dilemma_NN ._PUNC. </s>
You are looking at pair no 16
TOP
#P
VP
NP
RP
NP
<s> #NULL# _#NULL# mohr_X@ se_P4 svìøil_Vp pøíteli_N3 se_R7 svým_P7 trápením_N7 ._Z- </s>
You are looking at pair no 16

Figure 9. Radu Florian's Tree Viewer

Conversion between Dependency Structures and Constituent Structures

Once we had parses for the text, we used Lance Ramshaw's tools to convert these to sgml (see bin\tosgml.prl)

The English parsers had richer constituent structure than the Czech parses. The Collins parser was trained on constituents derived automatically from the Czech dependency trees.

Morphological Anchors

We then began looking for systematic correspondences, using the part of speech tags, for which there were relatively close affinities as well as a probabilistic bilingual word lexicon built in Prague from the Readers Digest Corpus, and began looking for ways to infer the Czech structure. The Czech data was tagged using the conventions set for the Prague Dependency Treebank. The English parsers were developed using the tags in the Penn Treebank. The following table truncates the tags on both sides and matches the basic parts of speech. We used these tag affinities to anchor nodes in the corresponding dependency trees.

Part of speech	**Czech tag**	**English tag**
adjective	A-	J-
adverb	D-	R-
conjunction	J-	CC, (IN)

determiner	--, (P-)	D-
existential (English “there is,” “there was”)	--	EX
interjection	I	INTJ
modal	(V-)	MD
noun	N-	N-
number	C-	CD
particle	T-	PRT
possessive	P-	POS, -$
preposition	R-, (J-)	IN
pronoun	P-	PRP
punctuation	Z-	PUNC
to (English word)	--, R-	TO
wh- word (who, how, etc)	P-	W-
verb	V-	V-
unknown, misc.	X-	F-, L-, S-, U-, etc.

Figure 10. Tag Affinities

A short sample of unigram frequencies for the part of speech tags is shown in Figure 11. These numbers were compiled from a set of aligned Reader’s Digest sentences. The set contained sentences with a combined length (English sentence length + Czech sentence length) of twenty or less. Notice that there is a similar distribution of these tags.

POS	**Czech**	**English**
Conjunction	433	231
Determiner	--	892
Modal	--	173
Particle	74	4
Preposition	658	684
Pronoun	1297	1262
Unknown	531	22
Total number of words	9317	11214

Figure 11. Unigram Frequencies of Tags for Function Words*.*

The unigram frequencies for all of the tags in the data set are shown in Figure 12. Notice the close correspondences for the major parts of speech - adjective, adverb, noun, preposition, pronoun, verb. The English sentences tend to be slightly longer than the Czech sentences. This is probably due to Czech's rich morphology; some words necessary in English are unnecessary in Czech. These results indicate a close match, word for word, between the Czech and English translations.

Part of speech	**Czech tag**	**Number of occurrences**	**English tag**	**Number of occurrences**
adjective	A-	655	J-	643
adverb	D-	883	R-	920
conjunction	J-	433	CC	231
determiner	--, (P-)	--	D-	892
existential (English "there is," "there was")	--	--	EX	33
interjection	I	3	INTJ	0
modal	(V-)	--	MD	173
noun	N-	2112	N-	2868
number	C-	197	CD	203
particle	T-	74	PRT	4
possessive	P-	[see pronoun]	POS, -$	75
preposition	R-	658	IN	684
pronoun	P-	1297	PRP	1262
punctuation	Z-	[removed]	PUNC	[removed]
to (English word)	--, R-	--	TO	211
wh- word (who, how, etc)	P-	[see pronoun]	W-	106
verb	V-	2420	V-	2442
unknown, misc.	X-	531	F-, L-, S-, U-, etc.	22
total number of words	**--**	**9317**	**--**	**11214**

Figure 12. Unigram Frequencies of All Tags*.*

Lexical Anchors

Although we found that the part of speech tags were quite reliable for finding matching structures, even when we did not have information about lexical correspondences, translations, etc, we did explore using lexical anchors to identify correspondences. The English translations shown beneath the Czech words in the parse were inserted automatically using the probabilistic lexicon built by Cmejrek and Curin in Prague. We simply chose the word that was listed as most probable.

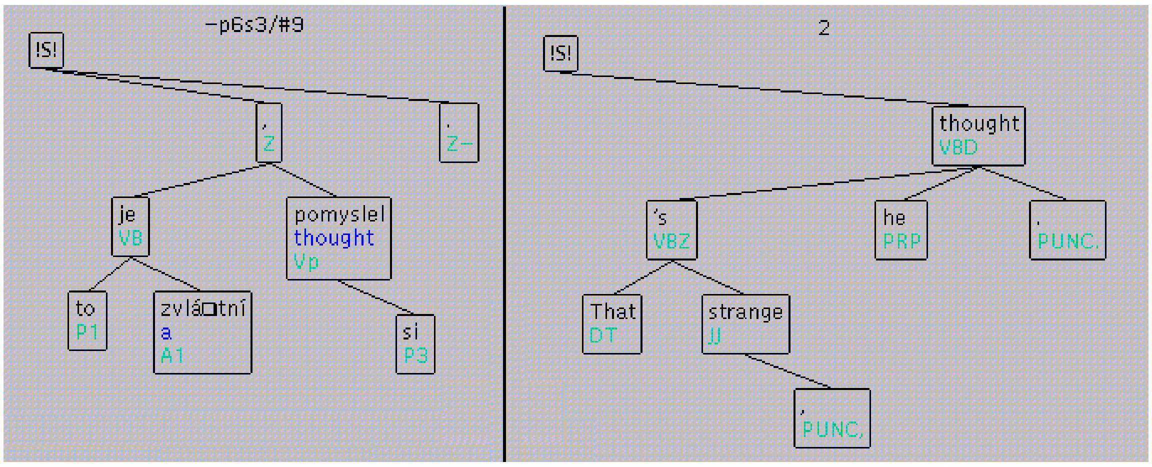


Figure 13. Lexical Anchors

The work by Cmejrek (1998) and Curin (1998) produced about 12,000 words with possible translations ranked in order of probability. Figure 14 shows the possible translations for *absurdní*, namely, *absurd*, *possible*, and *preposterous*. In this case, we are happy with the top choice for the lexical anchor.

absurdní	9	
3.636364e-01	4	absurd
1.818182e-01	54	possible
1.818182e-01	2	preposterous
9.090909e-02	23044	<EMPTY_WORD
9.090909e-02	7039	And

******** total 1.000000e+00 ***************

Figure 14. Probabilistic Lexicon

Improving Lexical Anchors

However, the top-ranked choice was not always the best one. When we looked at these closely, we thought of two ways to improve the translation choice: the first was to use the evidence from Minimum Edit Distance (Levinshtein Distance) to identify cognates. Figure 15 shows how we could pick *auction* as the translation for *auk\350n\355*, skipping past the more probable (but incorrect) *at*. (Thanks to Christoph Tillman for helping us with the code for measuring the Levinshtein Distance).

			Minimum Edit Distance
aukční	at	2.50	
	auction	0.71	*****
	sale	1.5	

sotheby's	1.0

Figure 15. Minimum Edit Distance to Improve Guess

The other idea was to use semantic evidence to identify related translation possibilities, using WordNet. In Figure 16, we could use this evidence to accept *absurd* and *preposterous* as translations for *absurdni* because both *absurd* and *preposterous* are in the same synset in WordNet v1.6. We could then exclude the incorrect translation possible. Of course in this instance, the top choice given was correct. Nevertheless, we thought using WordNet might be a fruitful exercise.

		Same synset (Wn 1.6)
absurdní	absurd	*****
	possible	
	preposterous	*****

Figure 16. Semantic Evidence from WordNet to Improve Guess

1. Conclusion and Future Work

As we mentioned in the abstract and introduction, this survey was preliminary and its primary purpose was educational. The Czech Readers Digest corpus is clearly a very valuable resource for research in a variety of areas, including parsing and the automatic construction of resources for machine translation.

2. Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. (#IIS-9732388), and was carried out at the 1998 Workshop on Language Engineering, Center for Language and Speech Processing, Johns Hopkins University. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation or The Johns Hopkins University.

3. References

1. [Charniak 1997]
   Eugene Charniak: *Statistical Techniques for Natural Language Parsing*. In: AI Magazine, Volume 18, No. 4. American Association for Artificial Intelligence, 1997.
2. [Cmejrek 1998]
   Martin Cmejrek: *Automatická extrakce dvojjazyčného pravděpodobnostního slovníku z paralelních textů* (Master's Thesis on automatic extraction. Univerzita Karlova, Praha 1998.
3. [Collins 1996]
   Michael Collins: *A New Statistical Parser Based on Bigram Lexical Dependencies*. In: Proceedings of the 34th Annual Meeting of the ACL, Santa Cruz 1996.
4. [Collins 1997]
   Michael Collins: *Three Generative, Lexicalised Models for Statistical Parsing*. In: Proceedings of the 35th Annual Meeting of the ACL, Madrid 1997.
5. [Curín 1998]
   Jan Curín: Master's Thesis on automatic extraction of bilingual terminology. Univerzita Karlova, Praha 1998.

6. [Zeman 1998 a]
   Daniel Zeman: *A Statistical Approach to Parsing of Czech*. In: Prague Bulletin of Mathematical Linguistics 69. Univerzita Karlova, Praha 1998.
7. [Hajic 1998]
   Jan Hajic: *Building a Syntactically Annotated Corpus: The Prague Dependency Treebank*. In: Issues of Valency and Meaning, pp. 106-132 Karolinum, Charles University Press, Prague, 1998.
8. [Hajic and Ribarov 1997]
   Jan Hajic, Kiril Ribarov: *Rule-Based Dependencies*. In: proceedings of MLnet Workshop on Empirical Learning of NLP Tasks. Pages 125-135.

Chapter 4: Structured Language Model Parsing

Oren Schwartz, Univ. of Pennsylvania

Introduction

The Structured Language Model (SLM) was introduced by Ciprian Chelba and Frederick Jelinek as a language model that uses a statistical parser in a left to right manner in order to exploit syntactic information for use in a language model as part of a speech recognition system (Chelba, Jelinek, 1998; Chelba, 1997). In traversing a sentence, the Structured Language Model produces a series of partial lexical parses whose exposed headwords are used instead of the previous two words in a trigram-like prediction process. It is conjectured that this language model could be easily modified to produce good complete parses for Czech. One of the attractive features of the parsing method used in this model is that it can be easily modified to handle "crossing" or non-projective dependencies, a feature of the Prage Dependency Treebank that our other parsers currently ignore. Chelba, who implementated his Structured Language Model in C++, was gracious enough to allow us access to his code in order to modify it for our purposes. During the workshop, with time constraints and the usual range of difficulties encountered, we had time only to modify this Structured Language Model to work properly with the Czech data, to experiment with unknown word statistics, and to use part of speech tags generated by a version of Jan Hajic's exponential model statistical tagger. The results obtained during the workshop are encouraging as they were obtained with a version of the SLM which, though modified, is still not optimized for parsing.

The Structured Language Model

The Structured Language Model consists of two main parts: a parser and a predictor. The model proceeds along a sentence in a left to right manner, constructing partial parses for the sentence prefix available at each word. These partial parses consist of binary branching lexical parse trees where each node is associated with a lexical item and a nonterminal or part of speech (POS) label. Each nonterminal label and lexical item pair that covers a partial parse is referred to as a headword. The predictor uses the last two exposed headwords over a sentence prefix to predict the next word in the sentence. With parameter reestimation Chelba and Jelinek report results that this technique does achieve significantly lower perplexities for the UPenn Treebank of Wall Street Journal text (Chelba, Jelinek, 1998).

Partial parses are constructed in a binary manner by considering the last two headwords and their associated tag information. The parser can choose from three moves at any given point: ADJOIN-RIGHT, ADJOIN-LEFT, or NULL. An ADJOIN-RIGHT move creates a new nonterminal that covers the last two words, percolating the right word up the parse as the headword of the new phrase. ADJOIN-LEFT acts similarly, except that the left word is percolated up. After each adjoin operation, headwords are renumbered. The parser continues to build syntactic structure over a sentence prefix in this manner until the most probable move is the NULL move, which does not change the parse structure, but passes control to the predictor, which then predicts the next word and its POS tag from the previous two headwords. The model proceeds down each sentence in this manner until the end of sentence marker is reached. Because of the large (exponential) number of possible parse trees associated with any given sentence, this model uses a multiple stack search with pruning through the space of sentence parse tree hypotheses (see Chelba, Jelinek 1998). In this manner, hypotheses with equal numbers of parser operations and predictions are compared against eachother.

Probability Model

The probability model used by the SLM was not changed during the course of this workshop. The probability of a sentence consists of two main parts, the predictor probabilities and the parser probabilities. For our purposes, a headword consists of a word and a nonterminal label, $h = \langle w, t \rangle$ where the nonterminal label may be a POS tag if the headword is a terminal item in the parse tree. The total probability of a sentence of length n and its associated parse tree, T_n, is the product of the predictor probabilities for each new word-tag pair, multiplied by the product of the probabilities of the sequence of moves taken by the parser at each position k along the sentence.

The predictor decomposes the probability of the current headword, h_0^k, given partial parse T_k of the k-word sentence prefix as follows. We first predict the current word given the previous two headwords for a particular partial parse of a k-word sentence prefix, and then the POS tag associated with it given those headwords and the word itself. The predictor probability is then

$$P\left(h_0^k \middle| h_{-2}^k, h_{-1}^k, T_k\right) = P\left(w_0^k \middle| h_{-2}^k, h_{-1}^k, T_k\right) \cdot P\left(t_0^k \middle| h_{-2}^k, h_{-1}^k, T_k, w_0^k\right)$$

.

The parser probability for the partial parse of the k-word sentence prefix given the partial parse of the (k-1)-word sentence prefix is the product of the probabilities of the sequence of actions of length N_k, ending with a NULL move:

$$P(T_k | T_{k-1}) = \prod_{i=0}^{N_k} P\left(a_i^k \middle| h_{-2}^i, h_{-1}^i, T_{k-1}, a_1^k \ldots a_{i-1}^k\right)$$

.

The total probability of a sentence is then

$$P(W, T) = \prod_{k=0}^{n} \left[P\left(h_0^k \middle| h_{-2}^k, h_{-1}^k, T_k\right) P(T_k | T_{k-1}) \right]$$

Dependencies and Binary Trees

Clearly, the parses constructed by the Structued Language Model are binary lexical parse trees. Given a lexical parse tree of any form, a corresponding dependency structure for the same sentence can be uniquely determined. The SLM is a statistical model and thus requires training data in the form of binary lexical parse trees. The correspondence from dependency structures to lexical parse trees for a given sentence is one to many, as neither nonterminal labels nor the number of intermiediate nonterminal phrases and their internal structure are uniquely defined by a dependency structure. As the Collins parser also works with lexical context-free trees, efforts were made by our team to find an optimal way to construct these trees from dependencies. For the Structured Language Model parsing effort, the same techniques used with the Collins parser were used with a simple modification to restrict these trees to be binary branching. The modification converts any rule with more than two children into a binary branching sequence of rules by introducing new nonterminals to cover the headword of the phrase and its left neighbor if it exists or its right neigbor otherwise. This is done recursively through the tree until each tree is uniformly binary branching. New nonterminals are constructed from the nonterminal label of the parent of the phrase by adding a prime symbol. The existence of unary constructions is allowed, provided that the child is a terminal. By using the same methods for converting dependencies to trees, we benefit from all improvements to these algorithms made during the workshop.

Unknown Words

The unknown word problem in Czech is substantial, due in large part to the heavily inflectional quality of the language. Approximately 12% of all tokens and 35% of all words encountered in the test data set were never seen in the training set. Because the SLM was developed as a language model, it has little use for out of vocabulary words, as they will be predicted infrequently. The first attempt to use the SLM for parsing Czech did not change the manner in which unknown words were dealt with, leading to a fairly low baseline result. To alleviate some of the problems caused by unknown words, we add a threshold in the training data such that if a word is seen less than that threshold number of times, it is considered unknown. Currently, all unknown words are mapped to a unique string, "<unk>" so that the statistics gathered for unknown words in training hold for those encountered in the test.

Threshold	Unknown words in test	Unknown tokens in test
none	35%	12%
3	57%	19%
5	67%	23%

Part of Speech (POS) Tags

Jan Hajic (Hajic, Hladka, 1998) brought to the workshop an exponential model statistical parser that performs at ~92% accuracy which was run on our test data without having seen any of it in training. These POS tags for each word in the test data were available to our parsers. By using these tags for unknown words we notice a substantial improvement over our baseline performance for SLM parsing. When we use these tags for all words, we notice a very small (<.3%) improvement over the results with these tags for unknown words only, suggesting that the SLM is doing well in assigning POS tags to words that it does recognize.

It should be noted that even though the part of speech information in the Prague Dependency Treebank is quite extensive, all results shown here were achieved with a drastically reduced tag set. If the full set of tags were to be used, a drastic sparse data problem would arise. The reduced tag set used here is the same one used in the Collins parser for this workshop. Again, further work in determining an optimal reduced set of tags may significantly improve performance here, as well as in the Collins parser.

Preliminary Results

Training was done on a set of 19,126 sentences. Two test sets were used, a development set with 3,697 sentences and an evaluation set with 3,807 sentences. The composition of these data sets are explained in the Data section.

Baseline results:

No unknown word statistics. No use of external (MDt = Machine Disambiguated tags) POS tags.

devel	54.7%
eval	55.5%

Use MDt tags for:

unknown word threshold:3	none	unk	all
devel	57.91%	67.42%	67.54%
eval	57.70%	67.16%	67.45%

unknown word threshlod:5	none	unk	all
devel	--	68.04%	68.18%
eval	57.29%	68.12%	68.32%

Further Research

There are many possible modifications to the algorithm and details of its implementation that have not yet been explored, each of which with the potential to improve the performance of the SLM in terms of the parsing effort. Some are specific to the SLM, while others are potentially beneficial to many parsers. Some of the more promising avenues are mentioned here.

Crossing Dependencies Modification

One of the interesting features of the sentences in the Prague Dependency Bank is that they contain crossing or non-projective dependencies. Fortunately, these crossing dependencies make up only about 2% of all dependencies, so for the most part handling these dependencies correctly was not a top priority for most of our parsing efforts. The SLM can be easily modified to produce crossing dependencies directly. The modification is as follows. The parser, instead of considering only the last exposed headword and the current headword as possibilities for an adjoin operation, now considers the current headword and the entire list of previous exposed headwords as possibilities for adjoin operations. It must now predict an action together with an integer specifying which previous headword should participate in the adjoin operation. Of course, we must condition these probabilities on certain features to penalize adjoins of headwords that are not adjascent, as these non-projective dependencies are only a small fraction of all dependencies. The exact nature of the features to condition on, and the details of the decomposition of the probability model for smoothing purposes are open questions.

Closing Parses

When the end of sentence marker is reached in the Structured Language Model, all partial parses are forced to ADJOIN-RIGHT with probability 1 to the end of sentence marker until the only two headwords are the end of sentence and beginning of sentence markers. When the SLM is used as a language model, once the end of the sentence has been predicted, there is no further use for its syntactic structure, and closing parses in this manner is perfectly acceptable. For our ends, however, it would clearly be beneficial to allow the statistics to guide the closing of a parse tree.

Direct Optimization for Parse Structure and Parameter Re-Estimation

The SLM currently uses the perplexity measurement as criteria for optimization in the reestimation of parameters. We would like to perhaps introduce a measure for optimization of the parse structures output directly. In so doing, we may be able to explore reestimation techniques with regard to the parses we create.

Predictor Probability Decomposition

Because the SLM was designed as a language model for use in continuous speech recognizers, the predictor first predicts the next word from the previous two headwords, and then the POS tag from that word and the previous two headwords (see Probability Model above). As we are now focusing on parsing, we might consider predicting the POS first, as this might be more consistent with recovering reliable parses for sentences with unknown words.

Data Annotation Problems

Certain aspects of the manner in which the Prague Dependency Treebank is annotated may be less than optimal for the purpose of producing a statistical parser for Czech sentences. For example, in the PDT coordinated phrases are dependent on the coordinator in the sentence. Certainly for predicting, one would assume that the headwords of the coordinated phrases are more likely to give a good estimate of what the next word would be than the coordinator. We might like to explore the effects of handling certain features (e.g. coordination, punctuation, etc.) in the Czech data in slightly different ways so that our parsers can make use of as much information as possible.

Nonterminal Labels and POS Tag Sets

It should be noted that the manner in which sentences are annotated with dependency structures and the manner in which those structures convert to trees are very important concerning the performance of our parsers. Nonterminal labels should ideally describe completely the terminals they cover to the rest of the sentence. Fully exploring the best choices for nonterminal labels and rule generation is a task which has yet to be completed, and one which may yield significant improvements in parser performance.

Likewise, finding an optimal set of POS Tags may also increase performance significantly. There was simply not enough time during the workshop to explore fully techniques for clustering these tags, but efforts were made and are explained elsewhere in this report.

References

Chelba, Ciprian and Frederick Jelinek, "Exploiting Syntactic Structure for Language Modeling." 1998.

Chelba, Ciprian, "A Structured Language Model," 1997.

Hajic, Jan and Barbora Hladka, "Tagging Inflective Languages: Prediction of Morphological Categories for a Rich, Structured Tagset," In: Proceedings of ACL/Coling'98, Montreal, Canada, Aug. 5-9, pp. 483-490, 1998.

Chapter 5: The Superparser

Eric Brill, Barbora Hladká

There has been a great deal of recent interest in classifier combination as a means for increasing classifier accuracy. We have already demonstrated (BrillACL98) that combining part of speech taggers can result in a significant accuracy improvement over any single tagger. Our group intended to develop multiple Czech parsers, therefore we planned to explore methods for combining these parsers.

Since we initially only had one successful parser, an adaptation of the Collins parser, we instead began by exploring methods for diversifying and combining a single parser. One of the most successful methods for doing this is known as boosting. Essentially, multiple learning iterations are carried out, where the classifier is trained, then applied to the training set. At each iteration, instances that are incorrectly classified are given a greater weight in the next iteration. By doing so, in each iteration the learner is forced to concentrate on instances it was unable to correctly classify in earlier iterations. In the end, all of the trained classifiers are combined via weighted voting.

The problem with applying boosting to the Collins parser is that the parser is very fast to train, but very slow to apply. Each iteration of boosting requires one training run and one application run. But applying the parser to a sufficiently large training set would take days, and hundreds of iterations are often required. Another method for diversifying a single classifier is bagging. In bagging, we get a family of classifiers by training on different portions of the training set. The method works as follows. We first create N training bags. A single training bag is obtained by taking a training set of size S and sampling this training set S times with replacement. Some training instances will occur multiple times in a bag, while others may not appear at all. Next, each bag is used to train a classifier. We now have N classifiers. These classifiers are then combined by simple voting.

Bagging is effective in cases where small variations in the training data result in significant differences in the resulting classifier. If training is relatively insensitive to small training data differences, then the N resulting classifiers will not be significantly different, and therefore combining these classifiers will not give any significant improvement over a single classifier.

For our first experiments on Czech, we took the Collins parser, which had been retrained for Czech. We took the Czech corpus training set and generated three training bags, which were then used to train three Collins parsers, one from each bag. Below we show the constituent accuracy as a function of the number of parsers that posited that constituent:

Number of Parsers	**Accuracy**
1	28.0
2	50.3
3	92.2

From this we can see that voting has promise, as the number of parsers agreeing on a constituent gives us a great deal of information as to the probability of that constituent being correct. Given this insight, we next went ahead and generated bagged parsers, combining their outputs as described above. Below are the results for these experiments:

	Precision	**Recall**	**P+R**
Original Parser	77.0	77.0	153.9

1 Bag	76.0	76.0	151.9
6 Bags	81.1	74.6	155.9
10	80.7	75.4	156.2
16	80.6	76.0	156.6

First, we note that a bagged parser by itself underperforms the original parser. This is because the original parser is trained on a "true" distribution of parse trees, whereas a bagged parser has a different sentence distribution. Also, the bagged parser was not exposed to all of the sentences in the training set. However, we see that when we combine a set of bags, we achieve performance significantly better than the original parser.

While there are many applications for which an imbalance of precision and recall is not problematic, there are also cases when we want a balanced parse, in other words a dependency structure that is complete, with every word having one and only one link pointing to it. To allow for such a structure, we changed our voting scheme. Instead of voting on a constituent basis, the parsers voted for every word W which link X --> W should be chosen.

	Precision	**Recall**	**P+R**
Original	77.0	77.0	153.9
16 bags unbalanced	80.6	76.0	156.6
16 bags balanced	77.8	77.8	155.6

We see that this method of combination, resulting in a balanced precision and recall, still gives us an improvement over the original parser, but not as great an improvement (in terms of P+R) as was achieved when voting was done on a constituent basis.

At the end of the workshop we had a number of parsers at our disposal for Czech, so we next explored methods for combining the outputs of those parsers. Below we show the oracle accuracy for various combinations of parsers, in other words the accuracy when an oracle can choose constituents from the union of all constituents posited by at least one parser.

	Oracle Accuracy
Original	77.0
+ 10 Bags	86.8
+ 18 Bags	88.1
+ 18 + Oren + Dan	91.7
+ Oren + Dan	86.6

Each parser does seem to add potential information. However, note that the Oren and Dan parser together give fewer good constituents than can simply be realized via bagging the original Collins parser.

Below are the results for constituent voting, where each parser has a weighted vote. The vote is weighted by the parser's estimated accuracy.

	P+R
18 Bags	156.6
18 Bags + Original	156.6

18 + Orig + Dan	156.6
18 + Orig + Oren	156.6
18 + Orig + Oren + Dan	157.2

From this we can see that at least using this simple combination scheme, the Oren and Dan parser do not provide us with useful information beyond what can be exploited from the Collins parser via bagging.

Below we show our final results. The method we chose for our final evaluation was to combine the 21 bags, without using the other parsers or the original parser.

	DeV			**EvaL**		
	P	**R**	**P+R**	**P**	**R**	**P+R**
Baseline	77.0	77.0	153.9	79.1	79.1	158.3
Unrestricted	80.6	76.0	156.6	81.8	79.1	160.9
Restricted	77.8	77.8	155.6	79.9	79.9	159.8

We also ran experiments on English, by bagging the Collins English parser.

	P	**R**	**P+R**
Original Parser	88.7	88.4	177.1
1 Bag	87.6	87.6	175.2
Original + 3 bags	90.5	87.1	177.6

References

Eric Brill and Jun Wu(1998). Classifiers Combination for Improved Lexical Disambiguation (COLING/ACL 1998).

Thomas G. Dietterich(1997). Machine-Learning Research: Four Current Directions. AI Magazine. Winter 1997, pp97-136.