Rule-based analytical parsing of Czech

Václav Klimeš

Abstract

This paper introduces two methods of dependency parsing by means of rules automatically inferred from training data. For Czech the accuracy of the proposed parsers is considerably behind that of the state-of-the-art parsers, but the paper brings a new method of inference of rules and applies transformation-based learning to parsing in a novel way. Moreover, the parsers use properties of Czech language in a very limited way and thus can easily be used for parsing another language. The paper also introduces a method of assigning so called analytical functions that outperforms that currently used.

1 Introduction

Syntactic analysis, usually called parsing, is a procedure building a syntactic tree structure out of a string of words and other symbols and thus specifying relations among these words. There are several reasons to parse a text. One of the obvious is that a parser can be the core of a grammar checker,—,a tool checking whether the occurrence of a word in a certain place of a sentence is plausible. A parser is useful in the task of speech recognition as well, since it can correct some bad decisions of the acoustic model employed in this task. Moreover, knowledge of the syntactic structure of a sentence helps in machine translation of the sentence into another language, because it gives a clue to the meaning of the sentence. Although especially for this task even deeper analysis is desirable, the syntactic structure achieved by a parser is halfway towards this deep-syntactic description. Last but not least, a parser can be employed by annotators specifying syntactic structures (which can be then used e.g. for training tools performing the mentioned tasks) to pre-annotate the data and thus to make their work easier.

To build a complete description on the syntactic layer there is also a need to assign a token its s-tag, which describes the function the token has in the sentence.

This paper describes two parsers we developed and the procedure assigning s-tags. Both parsers need an annotated corpus for creation of their model, which they then use for parsing. Although their target language is primarily Czech, at least the latter of them is intended to be independent on the language being processed.

Section 2 introduces data and tools used. Section 3 describes the evaluation method used for reporting parsing results. In section 4 there is a review of parsers of Czech. Sections 5 and 6 describe both methods the parsers are based on and Section 7 compares them. Section 8 introduces the procedure for s-tags assignment; finally Section 9 concludes the paper.

2 Resources used

2.1 The Prague Dependency Treebank and layers of its annotation

The Prague Dependency Treebank (PDT), currently in version 2.0 (Hajič et al., 2006), is a long-term research project, whose aim is a complex, linguistically motivated (manual) annotation of a small part of the Czech National Corpus.[1] The annotation is carried out at three layers: morphological, analytical

[1]http://ucnk.ff.cuni.cz

(surface syntax), and tectogrammatical. The Functional Generative Description theory (Sgall, Hajičová, and Panevová, 1986) has been the main guidance for the annotation principles and rules of PDT. A collection of newspapers, an economical weekly and a popular scientific magazine have been selected as the source textual material for the PDT.

On the **morphological** layer, the morphological lexical entry (represented by a **lemma**) and values of morphological categories (a **morphological tag**, shortly **m-tag**, i.e. the combination of person, number, tense, gender, verbal voice, ...) are assigned to each word.

Since there are as many as 13 morphological categories, the number of unique m-tags appearing in the corpus is over one thousand, which makes the data very sparse. For this reason we use the approach described in (Collins et al., 1999). The authors, having solved the described problem for the sake of training a parser as well, proposed a very good compromise (although in a rather ad hoc manner) between the sparseness of m-tags and their descriptive power. In general, they have reduced an m-tag to just two characters: the first of them being the part of speech, the second one being a case if applied (with nouns, adjectives, pronouns, numerals writtem in words, and prepositions), or a detailed part of speech otherwise. We refer to this approach as to the **two-letter m-tag**.

At the second and third (**analytical and tectogrammatical**) layers of the PDT, a sentence is represented as a rooted tree. Edges represent the relation of dependency (also called "immediate subordination" in some other theories) between two nodes: the governor and the dependent. Rather than 'dependent–governor', we usually denote a pair of adjacent nodes as 'child–parent', since not all the edges correspond to the relation of dependency in the linguistic sense,—,some of them have a rather technical character, e. g. edges adjacent to nodes representing punctuation marks. Based on this relation we will sometimes refer to a node's grandparent, siblings, ancestor (transitive closure of parent relation), offspring (transitive closure of child relation) and so on.

Every token (word, punctuation) from the original text becomes a node at the analytical (surface-syntactic) layer, shortly **a-layer**, of annotation and a label called an **analytical function**, shortly **s-tag**, is assigned to every node, describing the type of surface dependency relation of the node to its parent. The original word order position of the corresponding token is also kept as a separate attribute.

One language phenomenon occurring at this layer needs special attention. **Coordination** is a relation among several tokens in a sentence where none of them is subordinated to any of the remaining ones. This "horizontal" relation, opposing to the "vertical" relation of subordination, needs a special treatment in the corpus. An auxiliary token, usually a coordinating conjunction or a comma, represents all members of a coordination, which are attached to it as its children. When every member of coordination should be a child of a node, the node becomes the parent of the coordination node. Conversely, when every member of coordination should be the parent of a token, the token becomes a child of the coordination node. Members of coordination are distinguished from their common children by a special attribute `is_member`.

In this paper we are not interested in the tectogrammatical layer.

2.2 Training and test data

Since we carried out experiments with analytical parsing when PDT 2.0 had not yet been available, all results of our tools are reported as concerning PDT 1.0. Its data consist of 1,507,333 tokens in 98,263 sentences and are divided into a training set, a development test (d-test) set, and an evaluation test (e-test) set approximately in ratio 10:1:1. Whenever we report results on PDT 1.0, they were obtained using automatically disambiguated version of the m-layer of the whole data; this also holds true for other tools whose results are reported here. Our tools are trained on all the training data of PDT 1.0 annotated at a-layer and are tested on all its d-test data annotated at a-layer.

Only when all the experiments with analytical analysis had been done, PDT 2.0 have originated and we wanted to take performance of our tools using it. However, the experiments were not redone for PDT 2.0, we only ran parsing method 2 and determination of s-tags on PDT 2.0 with the settings which proved to be the best for PDT 1.0. The results are reported at the respective places together with those obtained using PDT 1.0. Whenever we report results on PDT 2.0, they were obtained using purely human-annotated data; this also holds true for other tools whose results are reported here. Our tools are trained on all the training data of PDT 2.0 annotated at a-layer and are tested on all its d-test data annotated at a-layer.

2.3 fnTBL

For the machine learning part of our tool, we chose the fnTBL toolkit (Ngai and Florian, 2001), the fast implementation of transformation-based learning mechanism (Brill, 1992). Although we consider our choice to be good, the toolkit is aimed at classification tasks only and is not capable to process tree structures. How we have overcome these drawbacks is described at the appropriate places.

The rules which the toolkit tries to learn are specified by **rule templates** which have to be designed manually before the learning can start. A rule template is a subset of the possible names of features together with the name of a feature which bears information about the class the sample belongs to (since it is possible to perform more classification tasks at once). A rule is an instance of a template: particular values are assigned to all the features. The rule is interpreted so that a sample belongs to a given class if the given features have the given values.

3 Evaluation

All evaluation in this work is done using test data, which are distinct from training data. One of the copies is human annotated, we take it as the measure of correctness, and we call it the **golden-standard annotation**. The other copy being the subject of comparison is called **test annotation**.

For an evaluation of our task, which is analytical dependency parsing, we use the most common metrics,—,(dependency) **accuracy**. It is defined as the number of correct dependencies, i.e. those connecting the same parent in test annotation as in the golden-standard annotation given the same child, divided by total number of dependencies in one of the annotations.[2] We will denote accuracy as A below.

For Method 1, when a parser may be unable to determine the parent of a node at all, we may be interested in other characteristics of its performance as well. We define **coverage**, C, as the ratio of the number of determined dependencies to all dependencies and **precision**, P, as the ratio of the number of correctly determined dependencies to all determined dependencies. Accuracy is defined the same way and it is the product of coverage and precision in this case.

4 Previous works

There are several parsers of Czech and to be able to compare our results with their we mention only those for which results on d-test data of PDT 1.0 are available.

These parsers fall into two groups: in one of them there are those parsers developed originally for English with its constituency-based structures, which are then converted into dependency structures; in the other group there are parsers working with dependencies internally. The existing parsers are described in Table 1 and their performance on PDT 1.0 and for some of them on PDT 2.0 is given as well.

[2]From the description of a-layer it follows that all annotations of the same data have to contain the same nodes and thus the same number of dependencies, which is equal to the number of nodes.

Author	**Short description and references**	**PDT 1.0**	**PDT 2.0**
Michael Collins	originally a constituency-based parser; based on lexicalized probabilistic context-free grammar (Collins et al., 1999)	82.6%	
Eugen Charniak	originally a constituency-based parser; inspired by maximum-entropy(Charniak, 2000)	84.3%	
Kiril Ribarov	dependency parser; perceptron-based (Ribarov, 2004)	about 72%	
Daniel Zeman	dependency parser; statistically modeling dependencies as word bigrams (Zeman, 2005)	74.7%	75.0%[3]
Tomáš Holan	dependency parser; based on push-down automaton and using genetic algorithm to learn weights of rules (Holan, 2005)	73.9%	75.3%
Ryan McDonald	dependency parser; searching for Maximum Spanning Tree (McDonald et al., 2005)	84.4%	
Keith Hall and Václav Novák	improvement of Charniak's parser by means of recovering non-projectivities (Hall and Novák, 2005)	85.0%	
Zdeněk Žabokrtský	dependency parser; hand-written rules tailored for Czech (unpublished)	75.2%	77.9%

Table 1: Description and performance of existing parsers

However, the most successful Czech parser originated as a combination of some of the existing ones (Zeman and Žabokrtský, 2005) and its accuracy on PDT 1.0 is 86.3%. Also Holan reports that a combination of his above-mentioned parser and his other two less successful parsers improves the accuracy to 76.9% on PDT 1.0 and 77.7% on PDT 2.0.

5 Method 1: Optimal context length

This section describes our early experiments with analytical parsing and attempts to develop a machine learning method based on human-readable rules. Although the method is usable only in parsing and possibly in some other areas where the linear context of a token can greatly help, and even in these areas it exhibits some deficiencies, it introduces an original approach to machine learning using rules, which builds its model on sequences of symbols. Besides, some ideas are used in a more successful transformation-based method described in Section 6.

5.1 The basis of the method

The model of this parsing method has the shape of rules describing dependencies among tokens in a sentence. The method is based on the idea that to be able to determine the parent of a token, we (usually) do not need to know the whole sentence but it is sufficient to know only an important information from the context of the token in question. Thus the first problems to be solved are which part of the information is important and what does "context" mean.

When considering the context of a token we work with linear surface structure of the sentence only, for simplicity neglecting the structure of the rising dependency tree. This approach cannot obviously profit from the knowledge of dependencies of the neighboring tokens, though on the other side it cannot get confused when the dependencies are not determined correctly. We define the context of a token as

[3]on automatically disambiguated data

a continuous sequence of tokens from the linear surface structure which include the token in question. For linguistic reasons we consider only those contexts in which the parent of the token is also included: when the possible parent is out of the context, we have almost no information about it, which is too little to decide whether this token is the parent. The algorithm itself determines the optimal length of the context.

Since a context can contain an unlimited number of tokens, only little information about tokens can be processed so that the computation can enter the memory. For this reason we use a two-letter m-tag as the only information about a token.

The issue of determining which of the nodes contained in a rule is the parent of a node in question has a rather technical character and we encode it as the relative distance from this node. As a special case, when the parent of a token is the root node of the sentence, zero is encoded as the distance and interpreted in the described way.

A rule thus contains the tag of the token in question, the tags of the tokens in its context in the order they appear in, and the distance of the parent. Technically, the first and the second entry are delimited with a colon and the second and third one with the "greater-than" sign. In the notation of the context several special symbols are defined as well: `'^^'` means the beginning of a sentence, `'$$'` denotes its end, and `'__'` stays for the position of the token in question. For example the rule `N2: N4 A2 __ > -2` means: when there occurs a noun in accusative followed by an adjective in genitive and by a noun in genitive, the parent of the noun in genitive is the noun in accusative.

5.2 Inference of rules

Having decided about all other substantial aspects of the method, we can decide now how rules will be inferred. Our idea was to develop a system where once a rule is applied to a configuration, no other rule will be applied to the same configuration, i.e. a single rule will decide about the parent of a token. For this reason the parsing procedure is simple:

- for every input sentence
  - for each of its tokens
    - go through the rules in the order in which they have been learnt and apply the first applicable rule

A rule is applicable when the tag of the token in question together with tags of the tokens in the context as recorded in the rule matches with the sample.

From the sentence symbolically written as `Db Vp N1` (the sequence of adverb, verb in present tense, and noun in nominative where the verb is the parent of both the other nodes) we can in an obvious manner infer the following rules.

```
Db: ^^ __ Vp N1 $$ > 1
Vp: ^^ Db __ N1 $$ > 0
N1: ^^ Db Vp __ $$ > -1
```

However, these rules are very specific and thus of little use and there is a need to find more general ones. For this reason we regard rules inferred in this way as our training samples and on their basis we find rules which are general as much as possible (i.e. having the shortest possible context), but which unambiguously determine distance to the parent. The rules are generated starting from those with shorter context (with the condition that the context recorded in a rule contains the parent which the rule determines) and when a rule cannot determine a parent unambiguously, i.e. when on some samples it is applied in a wrong way, it is split into several more specific rules by extending the context.

For simplicity, we consider only rules with continuous contexts. In addition there is the constraint that the context recorded in a rule has to be long enough to contain the parent which the rule determines,—,otherwise the rule would contain no information about the parent (except for its distance from the child).

The training procedure, based on the ideas above, is as follows.

- infer training samples from the training data
- delete those samples where the distance is ambiguous[4]
- NEXT_TAG: for every tag of a child
    - NEXT_LENGTH: for increasing context length starting from 1
        - from training samples infer the rules whose contexts have the current length and contain the parent
        - determine how many times every such rule would apply correctly and wrongly at the training samples
        - choose and write down the best rule, i.e. that which was applied the most times correctly and never wrong
        - when such a rule does not exist, go to NEXT_LENGTH[5]
        - delete samples which the best rule was applied to[6]
        - if no more training samples exist, go to NEXT_TAG

A single matter remains to be solved. The analytical structure is a tree; however, since the parents of tokens are determined independently, the output structure may contain cycles. We chose to solve this by unlinking one edge between parent and child in every cycle. The best criterion for which edge to unlink seems to choose that one corresponding to the rule which was applied (to the training data) the least times and which thus can be considered to be the least reliable. For this reason there is a need to store the number of applications of a rule together with each rule.

With this method we achieved $C = 80.2\%$, $P = 77.4\%$, and $A = 62.1\%$, having 600,981 rules (in the training data there are 1,224,076 dependencies).

5.3 Inference with exceptions

However, the system of rule inference is not optimal, since because of exceptions (and errors in annotation) a rule splits into a number of less general rules. This can lead to a lower coverage (and even precision, since the rules are less linguistically motivated). The phenomenon can be illustrated on the following example: because of the rule `A1: A1 J^ __ N1 > -1` (`A1` means an adjective in nominative, `N1` is a noun in nominative, `J^` is a coordinating conjunction), the rule `A1: __ N1 > 1` is split into rules `A1: Vc __ N1 > 1`, `A1: P6 __ N1 > 1` etc., while many other good rules, e.g. `A1: P3 __ N1 > 1`, are not inferred at all, because the corresponding configurations were not seen in the training data. To solve this problem it is sufficient to modify the training algorithm to find exceptions to the rule and place them in front of the given rule,—,in our example the correct order should be `A1: A1 J^ __ N1 > -1`, `A1: __ N1 > 1`; the parsing algorithm already has the desired property to stop when the first rule is applied and thus it can remain unchanged.

For the sake of our algorithm we have to formalize what an exception is. An exception to a rule is a rule which applies correctly or does not apply at all when the original rule would apply in a wrong way; and which does not apply when the original rule would apply correctly.

The training algorithm given below considers the formalization; it also allows for exceptions to an exception etc.

There is one more issue to be solved: to decide when to consider a rule to be correct, although with exceptions, and when to consider it to be wrong. We defined this criterion as a certain ratio of correct to wrong applications of a rule,—,when the actual ratio is equal to or greater than this threshold, exceptions

[4]For obvious reason, when there are two training samples differing only in the parents, no rule determining the parent correctly can be generated from them.

[5]for the rule to be split

[6]As during parsing, a sample is considered no more once the parent has been determined.

to the rule are generated, otherwise the rule possibly splits as described above. Experiments show that the best ratio is 2:1.

The new training procedure is as follows.

```
• infer training samples from the training data
• delete those samples where the distance is ambiguous
• NEXT_TAG: for every tag of a child
    • NEXT_LENGTH: for increasing length starting from 1
        • from training samples infer the rules whose contexts have the current length and contain the parent
        • delete rules not meeting the above requirements (this step is not applied for the first time)
        • determine how many times every such rule would apply correctly and wrongly at the training samples
        • choose the best rule, i.e. that which was applied correctly the most times and its correct-to-wrong ratio is above the threshold
        • when such a rule does not exist, go to NEXT_LENGTH
        • if the best rule applied wrongly at least once
            • mark training samples where the best rule applied correctly and where it applied wrongly
            • call recursively this functionality (except for the inference of training samples) on the marked samples[7]
        • write down the best rule
        • delete samples which the best rule was applied to
        • if no more training samples exist, go to NEXT_TAG
```

For the algorithm with implemented exceptions we achieved $C = 87.7\%$, $P = 79.5\%$, and $A = 69.7\%$ ($\Delta A = 7.6\%$), having 355,608 rules.

We can also state one more condition: an exception does not apply when the original rule does not apply; however, this stricter requirement leads to a greater number of rules and thus, for the reasons described above, to slightly worse results.

5.4 Connecting dangling nodes

When striving for the best accuracy we can try to find parents which our algorithm was not able to determine. For this reason we developed a simple back-off procedure which operates in the following manner. From the training data we selected a small part which now serves as held-out data and parsing is trained only at the rest of the training data. The held-out data are parsed with the obtained rules and from the tokens having no parent assigned statistics are gathered: for these tokens probabilities of such situations are computed in which the token has another token as its parent, when this possible parent occurs in the sentence. Child and parent tokens are characterized by their two-letter m-tags only; parent tokens are characterized also by the information whether they occur to the left or to the right of the child token.

After the parsing the analysis of tokens with non-determined parents runs as follows: from possible parents occurring in the sentence that one is chosen which has the highest probability recorded. From all the tokens with the same characterization only the closest one to its possible child can be chosen. In practice it also turns out that adding a penalty for a big distance between a child and its parent improves the results,—,for every position between them the recorded probability is reduced by 1/15.

[7] for inference of exceptions to the best rule

Results produced by this procedure are less reliable than those of the parsing procedure (see drop of precision below); therefore, when breaking cycles, lower weights are artificially assigned to edges established by this procedure.

After adding this simple procedure we achieved $C = 97.9\%$, $P = 74.9\%$, and $A = 73.3\%$ ($\Delta A = 3.6\%$).

5.5 Deletion of unreliable rules

It turns out that the rules which applied to the training data only once are too unreliable: the procedure described above can assign parents better than these rules. When these rules are deleted from the rule file and only the remaining rules are applied, the result is slightly better: $C = 98.4\%$, $P = 74.7\%$, $A = 73.6\%$ ($\Delta A = 0.3\%$) and these are the final numbers of this method. Besides, the number of rules greatly decreases to 65,659.

5.6 Possible improvements

If somebody wants to amend the accuracy, the possible way may be to increase the generalization of rules somehow and thus hopefully to raise the accuracy of the method. There are at least two approaches leading to this. One of them is not to define context as a sequence of tags but rather as an expression operating on such a sequence, e.g. regular expressions seem to be very suitable for this task (see (Zeman, 2001)). The other approach makes the rules express agreement between a child and its parent (and possibly other tokens in the context).

However, these ideas would require a careful implementation so as not to exceed a reasonable amount of computing memory, since the number of possible rules during computation would highly grow.

6 Method 2: Transformation-based classification

This parsing method is based on similar ideas as method 1 is; however, we did not implement our own algorithm inferring rules describing dependencies, but we left this task to fnTBL, one of the toolkits performing transformation-based learning (see Section 2.3). The result of this is a high possibility to affect the choice of information important for parsing, a lower amount of programming work and a higher accuracy of the parser.

6.1 The basis of the method

Similarly to method 1, the basic idea of parsing is to determine the parent of each token independently on other dependency edges[8] and to use only information from the context of the token for doing so. Unlike in case of method 1, we can easily design the shapes of contexts[9] and choose what information from the context we use.

Since fnTBL is capable to solve classification tasks only, there is a question how to transform the task of determination of the parent to a classification task. The answer can be that we can classify a token into a class identifying its parent. The identification should be unique, i.e. it should choose (at most) one token as a parent, but it should, because of reliability, also describe a possible parent by means of taking some of its properties into account,—,the more linguistically motivated the descriptions are, the higher reliability of resulting rules we can expect. However, the description of a possible parent cannot be *too* specific, since otherwise it could be impossible for a rule to ever find a token with the given properties. A compromising solution is needed.

[8]since fnTBL cannot process tree structures, we have no other option

[9]however, it is no more possible to have contexts of unlimited length

It is possible to use two-letter m-tags for the task of description. To make the description unique, we append to its beginning the distance of the parent from its child, considering only tokens with the same tag as the parent has. E.g. `-1N4` means "the previous noun in accusative" and `+2VB` means "the second verb in present or future tense to the right", with value `0` meaning the (technical) root of a sentence.[10] This way of identification also solves the problem that, unlike in case of the method 1, we cannot force the context in a rule to contain the parent and thus we know almost nothing about it when it is beyond the context.

6.2 Connecting dangling nodes and enforcing of trees

The fact that in the context of a rule the parent determined by this rule may be absent also implies that a rule can determine a non-existing parent. E.g. a rule can assign `-1VB` as the parent of the current token, however, there is no verb to the left of it. For this reason we developed a small program which reads the parsed file and the rule file and places dangling nodes: the effects of rules assigning a non-existing parent to a token are undone, so that the node gets the parent determined by the latest rule from those assigning existing parents.[11]

As by the previous method the resulting structure is not necessarily a tree. This problem is also solved by disconnecting one of the edges forming a cycle: the edge established by the latest rule is chosen and the effect of this rule is undone (i.e. the child on this edge will have the parent appointed by the last but one rule which applied to this child). It also helps to consider edges containing children which were originally dangling to be less reliable and thus prefer these edges for breaking when they occur in a cycle.

The described criterion for choice of an edge to be disconnected turns out to be the best. We also trained probabilities of dependencies between tokens (similarly to the method described in Section 5.4) on held-out data and used them when deciding which edge to disconnect, but there has been no significant difference in the result; we got similar results when we disconnected the longest edge (measured by the linear distance of the two nodes forming it). Other approaches, as choosing the edge selected by the oldest rule instead of the latest one or choosing edges randomly, were found hurting the results a bit.[12]

6.3 The design of rules

There are many ways of creating the initial analytical structure,—,we experimented with so called "umbrella shape" (each node hangs on the root of its sentence) and "left string" (each node hangs on the previous node and the first node hangs on the root); several others can be found in (Zeman, 2005). However, when using fnTBL, it is best to initialize its input data in such a way that the initial assignment of classes is as correct as possible: one cannot expect higher accuracy, but the training is faster and uses less memory. This is why we set an initial class of every token (deciding according to its two-letter m-tag) to that in which it occurs most frequently in the training data.[13] This way we reduced the memory requirements and thus it was possible to add other templates into the template file.[14] This improved the accuracy of the parser comparing to simple umbrella-shaped initialization.

When designing rule templates, we did not succeed to incorporate a lemma, which is the other type of information obtained from m-layer, into templates containing m-tags. Rules containing lemmas of

[10] Only about 3% of the resultant rules determined the second or even more distant token with the given tag as the parent of a token.

[11] Tests show that about 4% of nodes were dangling and over one tenth of them has been placed correctly by the program.

[12] The difference between the best and the worst from all the given approaches is $\Delta A = 0.3\%$.

[13] Some tiny programs performing this initial classification are incorporated in the toolkit.

[14] Memory requirements of the training phase were limiting factor during almost all our computations.

```
tag_0=N6 => par=-1R6
tag_0=A2 => par=+1N2
tag_0=A1 => par=+1N1
tag_0=A4 => par=+1N4
tag_0=N2 => par=-1R2
tag_0=N4 => par=-1R4
tag_0=Z: tag_1=J, => par=+1J,
tag_0=A6 => par=+1N6
tag_0=N2 tag_-1=N1 => par=-1N1
tag_0=Vf => par=-1VB
tag_0=Db => par=+1VB
tag_0=P4 => par=+1VB
```

Table 2: The best rules for Method 2

two tokens did not enter the memory; and sets of rules with a lemma of one token turned out to be less accurate than those containing only m-tags.[15]

Surprisingly it helped to consider not only continuous contexts, but also contexts with "holes", i.e. such that properties of a token are assigned by a rule, but the properties of a token between that token and the token whose parent is looked for are not assigned. However, it did not help to consider contexts where the position of a token with given properties was not fixed, e.g. when a rule states that a token with tag `N4` has to occur somewhere between the first and the fifth position to the right from the token whose parent is looked for. A possible explanation of this phenomenon is that there are situations in which we do not need to know about all nodes in a context in case we know about those which can affect the dependency edge in question. E.g. the rule `tag_0=Z: tag_1=P1 tag_3=Vp => par=+1Vp` was generated saying that a punctuation mark which is followed by a relative pronoun in nominative, by an unknown token, and by a verb in past tense should have this verb as its parent. The situation corresponds to the beginning of a relative clause and according to the training data the unknown token can be an adverb, a verb in conditional form, a noun or a pronoun,—,none of these possibilities can alter the dependency between the punctuation mark and the verb.

After numerous experiments with various rule templates we have achieved the best result with the template file using only the morphological tags which has been designed this way: the rule templates cover all the rules with a continuous context of length up to seven (including the current position) and besides they cover all the rules containing the current position and up to two positions in the maximal distance of five. The accuracy is 74.5%.

The list of a dozen of the most useful rules is in the Table 2.[16]

6.4 Relocation of nodes hanging on the root

When rules assign a non-root parent to a node, but the node was dangling or it was cut off its parent because of cycles in the structure (see Section 6.2) and there are no other rules proposing a suitable parent, the node gets as the last resort the root of the sentence as its parent. We know almost certainly that the node is misplaced, but obviously we know little or nothing about its parent. There is, however, a special situation when we do know something: when such a node causes non-projectivity of an edge. Although non-projectivities are allowed in Czech, they are not frequent (according to (Zeman, 2005), in PDT 1.0 there is about 1.9% of edges non-projective) and thus it is almost sure that the node should

[15]After removing lemmas, memory requirements decreased, so we could enlarge the context,—,and this larger context turned out to improve accuracy more than incorporating lemmas did.

[16]These rules are generated, indeed, with initialization of training data to umbrella shape, so as the whole information is in the rules.

be relocated to a position where it does not break projectivity. Even so, there are many possible parents, but it helps most to relocate the node so that it becomes a child of the parent node of the non-projective edge.

After the application of this procedure the accuracy increased to 74.7% ($\Delta A = 0.2\%$) and this is the final accuracy of the method 2 on PDT 1.0.

On PDT 2.0, the final accuracy on d-test data is 76.6%, on e-test data it is 76.4%. The e-test data were used only for purpose of this evaluation, which was performed only once.

6.5 Analysis of errors

In Table 3 we present statistics in how many per cent of cases the parser has assigned the correct parent to a node with the given two-letter m-tag. For every part of speech exhibiting morphemic cases, records for all its cases are summarized into one number for the sake of and only one letter identifying this part of speech is then mentioned instead.

Vc	95.9	C}	77.5	AX	66.8	J^	49.7
RF	95.7	Xx	73.9	R	66.6	Vs	49.3
PX	91.5	Z:	73.8	Db	65.4	Ve	22.2
A	91.2	Dg	73.5	J,	64.2	J*	0.0
Co	87.5	Vf	73.1	Vp	63.9	CX	0.0
P	83.9	C=	71.6	VB	62.7	V4	0.0
AC	82.5	Cv	71.6	RX	58.3		
C	81.6	NX	70.6	Vi	55.7		
N	78.9	TT	67.2	II	50.0		

Table 3: Accuracy of determination of parents

Similarly, the Table 4 collects statistics in how many per cent of cases the parser assigned a *child* correctly to a node with the given two-letter m-tag.

Co	100.0	J,	83.1	Vs	70.3	Cv	56.1
CX	100.0	Vp	81.2	C=	69.1	Z:	50.1
Ve	100.0	N	80.8	Vi	69.1	II	50.0
C}	91.7	P	73.9	AC	67.2	J*	50.0
R	90.1	Vf	73.3	A	66.6	Db	44.0
AX	89.6	J^	71.9	Dg	63.9	RX	0.0
C	85.5	NX	71.3	TT	63.4		
Xx	85.3	VB	70.7	root	60.3		

Table 4: Accuracy of determination of children

Figure 1 displays statistics in how many per cent of cases an edge of a given length is correct and how many per cent of edges have a given length.

Figure 2 presents distribution of distances between the determined and the correct parent of a node, if the two differ. Misplacements concerning the root are treated as special cases: 28.4% of misplaced nodes hang on the root; on the other hand 3.7% of misplaced nodes should hang there.

These statistics bring no surprise: for the parser it is the most difficult to place verbs (except for conditional form of the verb *být* (*to be*), where the criteria are simple), adverbs, "prepositional phrases", coordinations (represented by coordinating conjunctions `J^`) and dependent clauses (represented by subordinating conjunctions `J,`). On the other hand, what could be called "noun phrases" in broader sense is

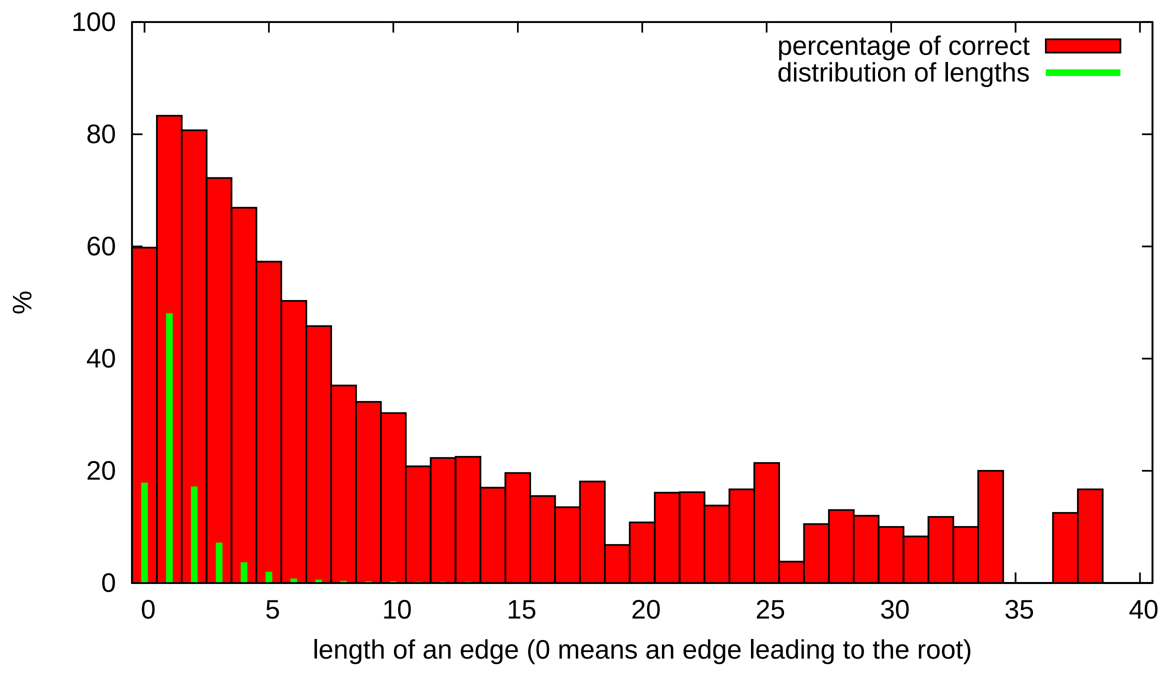

Figure 1: Percentage of correct edges with the given length and distribution of lengths of edges

usually correct: mostly adjectives, pronouns, and numerals expressed by words have the correct parent, and prepositions, nouns, and numerals expressed by words have correctly determined children. This conclusion corresponds to the statistics of accuracy of edges per their length: shorter edges, which probably correspond to "noun phrases", are more accurate.

6.6 Possible improvements

When we allowed rules to contain lemmas, the memory requirements increased highly. However, a lemma can bear valuable information, and this is why incorporation of lemmas to rules can improve accuracy. Obviously, the number of lemmas has to be reduced for the computation can enter the memory. There are at least two ways of doing so: to leave only the most frequent rules and to replace the others with a value meaning "other"; the other way is to create word classes on the basis of similar contexts, e.g. on the basis of bigrams. We did not test the former option, but made some experiments with the latter one. Alas, since Czech has a relatively free word order, the number of possible bigrams of lemmas is high and during computation only a few per cent of the training data could enter the memory. For the same reason the output did not look reasonably, e.g. lemmas with different part of speech were very often assigned the same class. It is possible to solve or at least reduce both these problems by performing a separate computation for every part of speech; however, this is certainly not optimal, since mixing different parts of speech in one class is sometimes well-founded and desirable. However, as regards incorporation of lemmas whose number will be reduced some way or another, we do not suppose it can help: during this operation some information, which we expect to improve accuracy, is inevitably lost.

A possible way to improve accuracy can be to divide the training into more phases. We train rules on one part of the training data and then on the second part we detect which edges are determined in the most reliable way, based e.g. on m-tags of nodes constituting them, or on their lengths (see Tables 3 and 4 and Figure 1). Then we create new data for training of unreliable edges only, since reliable ones are retained from the first phase of training. The advantage is that in the second phase the training data can be enriched with some features describing the partly created structure, e.g. with that telling how

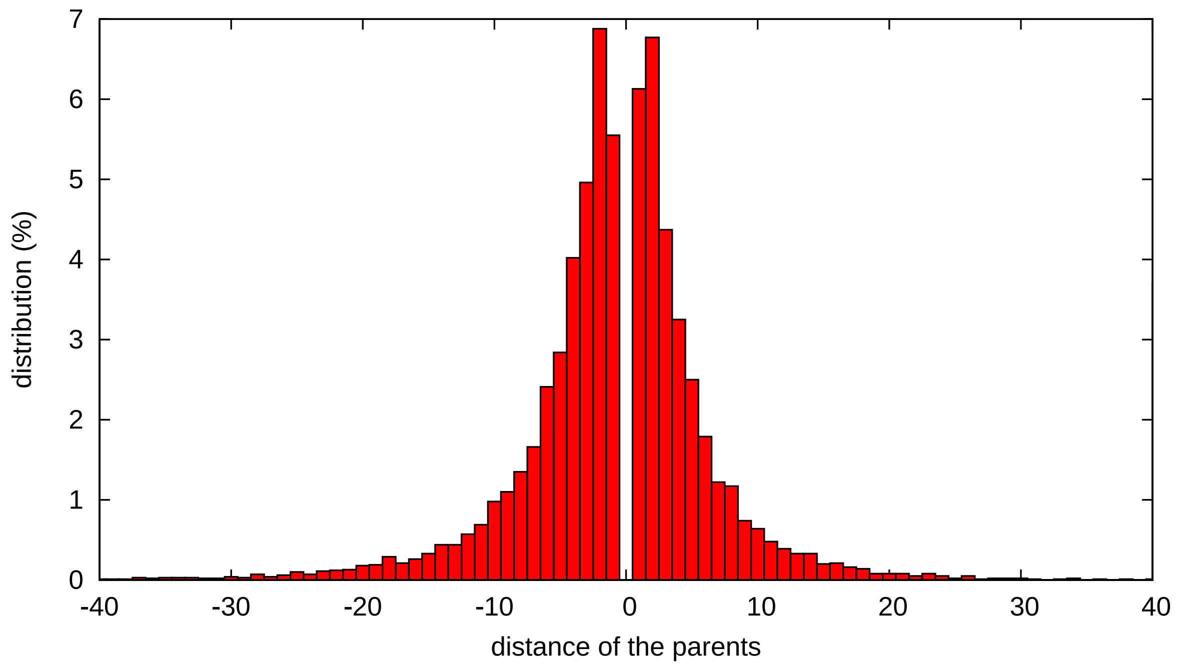

Figure 2: Percentage of non-zero distances between the determined and the correct parent of a node

many children a node has after the first phase; rules derived in this phase can profit from this additional and relatively reliable information.

It also may be worth trying to handle coordinations in another way than up to now, since coordinations, unlike the rest of edges, do not express dependency relation (see Section 2.1) and parsers are usually confused by this fact (only half of coordinating conjunctions is placed correctly, see Table 3). A possible approach is as follows. The training data are altered in the way that when a node has a coordination node as its parent, its effective parent is used instead (in case of common modifications, the closer effective parent is used); moreover, there is no need to generate rules determining parents of coordination nodes. When input data are parsed using rules obtained on these training data, coordinations have to be taken into account again. First of all the correct position of a coordination node has to be found: the "oldest" node such that the coordination node causes no non-projectivity when it becomes its child is chosen. Next, members of a coordination are determined: they are the closest left and right siblings of the coordination node; they become children of the coordination node and their `is_member` is set. After this step, if there is a punctuation as the closest left sibling of the coordination node, it and its left sibling become children of the coordination node as well and `is_member` of the left sibling is set; this step is repeated as many times as possible. There are two possible sources of errors (beside those caused by mistakes in parsing): when there are nested coordinations, it is impossible to determine which coordination node is the parent of the other; and common modifications of members of a coordination are determined as modifications of only one of them.

The low number of non-projectivities in Czech can also be employed as a constraint helping to improve accuracy, when applied to parsed data. We described our first successful attempt in Section 6.4, but obviously much more can be done in this area. (Zeman, 2005) classified non-projectivities of Czech from the linguistic point of view, but his classification is preliminary, constraints based on it are hard-coded and specific for Czech, and he employed just a small part of the whole potential. It may be worth trying to infer the constraints by a method of machine learning and to employ them automatically for identifying and repairing non-projectivities, where the correct structures are projective. Note, however, that this task is almost reverse to that described in (Hall and Novák, 2005),—,there the authors aimed to recover non-projectivities on places where they should be.[17] Their features seem to be reasonable even for the described task, but for this task we suggest their enriching with information whether the root of the subtree causing non-projectivity of an edge depends on the root of the sentence and on the fact whether this root is an ancestor of the nodes forming the edge, or an offspring of their ancestor.

7 The comparison of the parsing methods

The accuracy of Method 2 is a bit higher than that of Method 1 (73.6% versus 74.7%); however, this difference is partly caused by different memory requirements of the methods: while Method 1 uses almost 2 GB of memory, Method 2 uses all the available memory, which is 2.7 GB odd, and attempts to design rule templates which use approximately the same amount of memory as Method 1 does result to accuracy several tenths of per cent lower. Thus we can say that the performance of both methods is almost the same, although the methods are similar only to some extent. One possible explanation of this phenomenon is that for the results only those features which are the same in both methods are crucial; those in which the methods differ are not substantial. If this is true, the accuracy is affected by the choice of features of tokens posturing in rules and by the choice to determine parents independently; it is not affected by the exact method of inference of rules and of parsing. From this it would follow that to achieve better accuracy, the former properties of the algorithm have to be improved.

8 Determination of s-tags

To have a complete analytical structure as it exists in PDT there is, however, a need to assign an s-tag to every node. From the technical point of view this task is much simpler than parsing, since it is a mere classification task, perfectly suitable for e.g. the fnTBL toolkit which we used for parsing.

8.1 Previous works

To our knowledge the only tool assigning s-tags exists; it was created by Zdeněk Žabokrtský and has never been published, although it was often used by annotators. He solves the same problem using the C4.5 tool, which performs classification by means of construction of decision trees. As input features he uses the lemma (distinguishing only the 100 most frequent ones; the rest is replaced by a universal value meaning “other”), the part of speech, the detailed part of speech, the case, and the voice of a node whose s-tag is being assigned, those of its parent, and, if its parent is not an autosemantic word, also those of other its “youngest” ancestors that are autosemantic words. He also uses a number of children of the node in question, distinguishing only values 0, 1, 2, and more. He reports the accuracy of his tool to be 92.5–92.8%, when trained and tested on the appropriate data from PDT 1.0.

8.2 Design of rule templates

We continue his work and use fnTBL for the same task. We use the two-letter m-tag and the lemma of the node in question, its parent, its grandparent and its left and right tree siblings as input features of the toolkit; we also use the number of children of the node in question with values 0, 1 and more. These 11 features are combined in the following way: a rule can contain up to four features and up to two of them can be lemmas, but when there are exactly four features, only one of them can be a lemma. Moreover, at least one of the features has to refer to the node in question. The initial assignment of classes is done in such a way that the s-tag occurring most frequently in the training data is assigned to a node on the

[17]Although not mentioned explicitly, their method is able to correct all local misplacements, not only those caused by improper handling on non-projective edges.

basis of the combination of its two-letter m-tag and that of its parent.[18] Several of the most useful rules, only correcting the initial assignment, are given in Table 5.

```
tag=Z: child=2 => afun=Coord
lemma=" => afun=AuxG
tag=J^ child=0 => afun=AuxZ
lemma=, child=0 => afun=AuxX
lemma=) => afun=AuxG
tag=N6 ptag=R6 gtag=N1 => afun=Atr
lemma=( child=0 => afun=AuxG
tag=P4 lemma=on => afun=Obj
tag=N4 ptag=R4 gtag=N1 => afun=Atr
tag=N7 gtag=Vp => afun=Adv
tag=Vf plemma=být => afun=Sb
tag=N4 ptag=R4 gtag=N2 => afun=Atr
tag=N7 gtag=VB => afun=Adv
lemma=( child=2 => afun=Apos
tag=N1 ptag=Z: gtag=VB => afun=Sb
tag=Vp plemma=že => afun=Obj
```

Table 5: Most useful rules for s-tag assignment

Using the same data as Zdeněk Žabokrtský, we achieved accuracy 93.8%; on PDT 2.0 we achieved 95.1% on d-test data and 95.0% on e-test data.[19]

8.3 Attempt at an improvement: Two-phase training

Although in the templates described above it is possible to use features not only of the node in question but also those of its tree neighbors as far as we make them to be features of the node in question, it is impossible to use those features which are being determined; only features known before the training can be used. For our task this means that when determining the s-tag of a node, we cannot use s-tags of its neighbors.

For this reason we tried to split the training into two phases. In the first phase rules were trained as written above, but only on a part of the data. Using these rules, s-tags were assigned to the rest of the training data. Now features of nodes in these data were enriched with s-tags of its neighbors and based on these data, rules were trained correcting assigned s-tags.

Several template sets were tested in the second phase, starting from those referring to s-tags only and ending with those using the whole information as the templates in the first phase plus s-tags. There were also two ratios which were the training data split in. Although early experiments with this approach (using much simpler template sets in the first phase, and thus having a better chance to correct s-tags) improved the accuracy in about 0.3%, with the final template set there was, in the best cases, no improvement.

[18]Even this baseline assignment gives accuracy 81.7%.

[19]It should be said that in PDT 1.0 s-tags consist of two parts: the first part expresses what function the appropriate token has in a sentence; the second part expresses whether the token is a part of coordination and thus it supplies the attribute `is_member` missing in PDT 1.0. All the experiments of Zdeněk Žabokrtský and of mine are performed with this second part cut off and thus they are comparable.

9 Conclusion

However small the improvement of s-tag assignment over the previous tool seems to be, for those very high accuracies it corresponds to an indispensable 14% error reduction.

Method 1 has serious drawbacks, since rules produced by it can hardly be enriched with another type of information, e.g. with a lemma. Moreover, the method is hardly configurable,—,e.g. one cannot make a trade-off between accuracy on one hand and speed or memory requirements on the other hand. Despite its drawbacks, many ideas the method is based on are used in a more successful and very flexible Method 2.

Although the performance of both parsers is about 10% below the state-of-the-art parsers, both methods the parsers are based on are novel,—,the first one is completely new; to our knowledge, the second approach, rule-based classification, has never been used for dependency parsing. Improvements of the more promising of the parsers are suggested as well.

The parser is independent on the processed language: when one wants to adapt the parser for processing other language than Czech, in general only the choice of relevant positions of m-tags needs to be redone, the choice of rule templates may be altered, and, for languages allowing more non-dependencies than Czech does, usage of the respective post-processing procedure should be reconsidered,—,and, of course, the parser would have to be retrained. This is exactly the way how we developed the first version of a fully functional parser of a PDT-like annotated Latin corpus in no more than 10 minutes.

Acknowledgement

This research was supported by the grant of the Grant Agency of the Czech Republic No. 405/03/0913, the project of the MŠMT ČR No. LN00A063, and the grant of the Grant Agency of Czech Academy of Sciences No. T10147016.

References

Brill, Eric. 1992. A simple rule-based part-of-speech tagger. In *Proceedings of ANLP-92, 3rd Conference on Applied Natural Language Processing*, pages 152–155, Trento, Italy.

Charniak, Eugene. 2000. A maximum-entropy-inspired parser. In *Proceedings of NAACL*, Seattle, Washington.

Collins, Michael, Jan Hajič, Lance Ramshaw, and Christopher Tillmann. 1999. A statistical parser for czech. In *Proceedings of the 37th Annual Meeting of the ACL*, College Park, Maryland.

Hajič, Jan, Eva Hajičová, Jaroslava Hlaváčová, Václav Klimeš, Jiří Mírovský, Petr Pajas, Jan Štěpánek, Barbora Vidová Hladká, and Zdeněk Žabokrtský. 2006. Prague Dependency Treebank 2.0. CDROM. In press.

Hall, Keith and Václav Novák. 2005. Corrective modeling for non-projective dependency parsing. In *Proceedings of the International Workshop on Parsing Technologies (IWPT)*, Vancouver, British Columbia. Association for Computational Linguistics.

Holan, Tomáš. 2005. Genetické učení závislostních analyzátorů [Genetic-based Learning of Dependency Parsers]. In Peter Vojtáš, editor, *Sborník semináře [Proceedings of] ITAT 2005*, pages 47–54, Košice, Slovakia. UPJŠ.

McDonald, Ryan, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the Human Language Technology / Empirical Methods in Natural Language Processing conference (HLT-EMNLP)*, Vancouver, British Columbia. Association for Computational Linguistics.

Ngai, Grace and Radu Florian. 2001. Transformation-based learning in the fast lane. In *Proceedings of NAACL 2001*, pages 40–47, Pittsburgh, PA.

Ribarov, Kiril. 2004. *Automatic Building of a Dependency Tree—The Rule-Based Approach and Beyond*. Ph.D. thesis, Prague, Czech Republic.

Sgall, Petr, Eva Hajičová, and Jarmila Panevová. 1986. *The Meaning of the Sentence in Its Semantic and Pragmatic Aspects*. Reidel Publishing Company and Academia, Dordrecht and Prague.

Zeman, Daniel. 2001. How Much Will a RE-based Preprocessor Help a Statistical Parser? In *Proceedings of the 7th International Workshop on Parsing Technologies*, Beijing Daxue, Beijing, China. Tsinghua University Press. LN00A063.

Zeman, Daniel. 2005. *Parsing with a Statistical Dependency Model*. Ph.D. thesis, Prague, Czech Republic.

Zeman, Daniel and Zdeněk Žabokrtský. 2005. Improving parsing accuracy by combining diverse dependency parsers. In *Proceedings of the International Workshop on Parsing Technologies (IWPT 2005)*, Vancouver, British Columbia. Association for Computational Linguistics.