HOW MUCH WILL A RE-BASED PREPROCESSOR HELP A STATISTICAL PARSER?

Daniel Zeman
Centrum komputační lingvistiky
Univerzita Karlova
Malostranské náměstí 25, CZ-11800 Praha, Czechia
zeman@ufal.ms.mff.cuni.cz

Abstract
The present paper describes experiments on combining a statistical parser with a preprocessor built of regular expressions. While the syntactic structure of a sentence may be very complex and will never be fully covered by such simple machine, there are phenomena such as noun phrases and simple coordinations that do quite well. Even these "chunks" may theoretically be unlimitedly complex but we show that in reality they rarely do. So a shallow parser based on regular expressions can be built to preprocess the input text and solve the simple chunks without introducing too many errors. We discuss one implementation of such preprocessor that is very easy to write and covers roughly 20% of input words with an accuracy of over 90%. Then we describe two ways of combining the preprocessor with a parser and show that the performance of the parser improves both in speed and accuracy.

1. Introduction

The goal of the work reported on in this paper was to improve performance of a statistical parser. The idea was that there were very simple phenomena in the language that could be captured by a simple tool — where the second "simple" concerns programming effort as well as computational complexity. Combining such a tool with the parser would eventually speed up the parsing process. Moreover, the hope was that it would increase the parsing precision as well because the precision of the preprocessor was pretty high. The question was how much the preprocessor would increase the overall precision: due to its low recall it was expectable that most of structures it would recognize would also be found by the parser.

There is a body of related work on chunking but due to the limited space it cannot be thoroughly discussed here. So we only mention Nenadić (1998, 2000), seeming to us to be most close to our approach (see later).

The parser works in a dependency framework. Its task is, for each input word to find another word from the same sentence that is its governor. Thus the dependency trees produced by the parser contain no nonterminals; rather, every node except of the root corresponds to a word of the sentence. There is no grammar (context-free or other) generating directly the dependency trees. While parsers employing probabilistic context-free grammars can be used only indirectly in this framework (see Collins et al., 1999, for an example), there is a more straightforward way similar to n-gram language modeling. It differs from the n-grams in that it collects statistics about pairs of dependent nodes rather than n-tuples of neighboring words. The parsed language material is Czech; our data come from the Prague Dependency Treebank (PDT, see Böhmová et al.).

The preprocessor is implemented fully as a set of regular expressions in Perl[1]. While the parser can be trained on any language (as long as there is a dependency treebank), the preprocessor benefits from some properties of Czech (but it can be applied to other similar – e.g. Slavic – languages). It exploits the fact that the governing and the dependent node often neighbor in the sentence and agree in gender, number, case etc. This property simplifies recognizing of base noun phrases and other patterns.

2. The Preprocessor

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

[1] Nevertheless, the regular expressions are applied in a way that the resulting machine may be context-free rather than finite state.

The preprocessor is written in Perl (see Wall et al., 1996). There is a procedure that scans the corpus and repeatedly applies a set of regular expressions (RE's) to each sentence. The RE's can be easily modified and new RE's can be added. Every RE describes a local configuration that signalizes a dependency between two chunks of the input. A chunk is a word or a sequence of chunks that has been recognized by a previous application of an RE (and replaced by a representative word).

For instance, a coordination of adjectives is defined by the following three sub expressions:

```
"<t>A.([^X])([^X])(\\d)"
"<l>(a|i|nebo|ani)<"
"<t>A.\\1\\2\\3"
```

The Perl procedure combines these sub expressions into one RE. The resulting RE finds a sequence of words such that the description of the first word contains a match of the first sub-RE, the second word description matches the second sub-RE and so on. So in this case we are looking for a sequence of three words where the first one has a morphological tag beginning with A (adjective), having any character at second position (sub-part-of-speech), having known gender and number (unknown is encoded as X, which is prohibited by the RE), and having known case encoded as a digit. The second word must have the lemma *a* ("and"), *i* ("as well as"), *nebo* ("or") or *ani* ("nor"). The third word must again be an adjective (the tag beginning with A) and must agree with the first one in gender, number and case. The agreement is enforced by the sub-RE's \1, \2, and \3. They refer to the first, the second and the third parenthesized sub-RE's, respectively. In our case all referred parentheses occurred in the first word's tag, marking its gender, number and case.

The example RE would thus match for instance the coordination *přímým i nepřímým* ("direct as well as indirect"; instrumental case) that has the following (simplified) representation in PDT:

```
<f>přímým<l>přímý<t>AAMS7---A1-----
<f>i<l>i<t>J^-------------
<f>nepřímým<l>přímý<t>AAMS7---N1-----
```

On the other hand, the RE would not match the sequence *žlutého a černou* ("yellow / Masc. and black / Fem.") because the adjectives do not agree in gender:

```
<f>žlutého<l>žlutý<t>AAMS4---A1-----
<f>a<l>a<t>J^-------------
<f>černou<l>černý<t>AAFS4---A1-----
```

Two special parameters are passed to the procedure. They are interpreted as relative indices of the phrase head and of the phrase representative (1 and 2 in our case, the first word has the index of zero).

Each RE is applied repeatedly to each sentence until all matches are found. This can also accommodate some recursive constructions. Suppose we have an RE for adjective-noun NP's. And suppose there is a noun preceded by two adjectives that both modify it, e.g. *velký zelený strom* ("big green tree"). First the sequence *zelený strom* is recognized and replaced by the representative noun *strom*. This way the words *velký* and *strom* become neighbors and the sequence *zelený strom* can be recognized in the next run.

To keep the process simple, the order in which different RE's are applied is fixed. Nevertheless some RE's can be tried at several points to see whether the replacements created new material for them. To illustrate this, let's have two RE's called AN (adjective-noun NP), and NaN (coordination of nouns), and let's apply them in the order NaN-AN-NaN. Then if the input contains the sequence *čeští sportovci a slovenští sportovci a umělci* ("Czech sportsmen and Slovak sportsmen and artists"), the first NaN reduces it to *čeští sportovci a slovenští umělci*, this sequence is further reduced by the AN to *sportovci a umělci*, which is finally recognized by the second NaN. This example shows that the covered constructions can be richer than one might think. On the other hand it also reveals the main weakness of the approach: in other configurations such as *čeští hráči i Němci* ("Czech players as well as the Germans"), the RE sequence AN-NaN would be appropriate. So the method can be relatively successful only if there exists an ordering of RE's that matches the data in much more cases than any other ordering.

The programming simplicity and efficiency is a significant issue. The driving procedure was finished in a few days, a regular expression can be formulated and added in a minute or two and the Perl interpreter applies it to a corpus of 19126 sentences often in less then one minute[2]. On the other hand it is obvious that one can never cover the language this way, partly because a human cannot think of all possible configurations of the data, and partly because adding more complex patterns would make the expressions unintelligible and unmaintainable. That's why the preprocessor can only *help* the parser, not replace it.

[2] On a 266 MHz PC running Linux.

3. Combining the Regular Expressions with the Parser

There are two or three ways how to combine the two parsing tools. The first one assumes that the preprocessor would replace each recognized phrase by a representative word (typically by its head but coordinations – whose head is a conjunction – should be represented by one of their members, converted to plural). The parser then would see the preprocessed phrases neither during the training phase, nor during testing. Such phrases become atomic items for the parser.

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

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

4. Results

We ran several experiments to evaluate the finite state tool and its contribution to the performance of the parser. For each regular expression, we figured out its precision (measured as the number of correctly proposed dependencies divided by the total number of proposed dependencies). The test was done on 19126 sentences of the Prague Dependency Treebank (Böhmová et al.); this portion contains 346719 words (tokens). The following table shows precision of seven different regular expressions we tested (details on these RE's can be found in Zeman (2001)).

Expression	AaA	NaN	AN	DA	NgNg	NNg	RN
Precision	99.2	95.0	98.8	86.8	91.9		95.7

We tested all described RE's in the following order (note that NaN was applied twice):
`DA-AaA-AN-NaN-NgNg-NNg-NaN-RN`

Also this combined RE tool was tested on the same corpus. It proposed 69262 correct dependencies and 4424 false ones while leaving 273033 tokens unresolved. So the recall is 20.0 %, which means that the preprocessor is able to (correctly) help in one fifth of cases. The precision of the preprocessor is 94.0 %; only 1.3 % of all dependencies are errors that the parser cannot repair.

The above results are for data where morphological tags were assigned manually. Such evaluation is important because it shows the power of the RE's without biasing it by errors of other components. However, an application will hardly have manually annotated data available, so we are adding two other statistics. First, the RE set was run on tags and lemmas assigned by a statistical tagger (Hajič and Hladká, 1998). Every `<t>` in RE's was automatically replaced by `<MDt[^>]*>`; the `<l>` markups were treated similarly. The resulting machine proposed 60093 good dependencies and 5001 bad dependencies, 281625 remained unresolved; the precision was 92.3 %, the recall 17.3 %. The decrease in both the precision and the recall can be explained by the errors the tagger does.

The last test used ambiguous morphological input. Every `<t>` in RE's was automatically replaced by `<MMt>`; the `<l>` markups were treated similarly. The resulting machine proposed 69566 good dependencies and 8146 bad dependencies, 269007 remained unresolved; the precision was 89.5 %, the recall 20.1 %. The preprocessor had all morphological hypotheses available, so the recall slightly increased. But occasionally it combined two compatible hypotheses that occurred at neighboring positions but were wrong. That explains the drop in precision.

Finally we combined the finite state tool with the parser in all the ways described in Section 3. The resulting systems were tested on a part of the PDT that had not been used to train the parser. This testing portion consisted of 3697 sentences. Every word is now assigned a dependency so the precision is equal to the recall (and we call the measure *accuracy*). All experiments used machine-disambiguated morphology. The following table shows the baseline performance of the parser itself, the performance of RE's where unresolved words would always be attached to the root (labeled "sole RE's"), and the performances of parser with RE transparent preprocessor, non-transparent preprocessor and the postprocessor. Note also the times in the second line: they demonstrate how the parsing speed increases when some chunks are preprocessed.

	Parser	Sole RE's	Transp. Prepr.	Nontr. Prepr.	Postprocessed
Accuracy (%)	53.7	30.6	55.6	57.2	56.6
Time (min)	53	less than 1	30	46	54

The speed improvement is invariant but one may ask how significant is the accuracy improvement when the overall parser accuracy is quite low. So we also tested our RE's in combination with a different parser (it's a variant of the parser described in Collins 1999). This other parser performs better but we cannot modify its internals so we only could use the RE tool as a postprocessor. Even here the RE's helped:

	Parser	Postprocessed
Accuracy (%)	66.3	68.0

5. Conclusion

We showed that a set of regular expressions helps a statistical parser both in speed and accuracy. While the REs' contribution to the accuracy is to be expected to decrease as the parser itself improves, it hardly ever will damage the results because the precision of the REs is very high – higher than of any known parser for Czech or English. At the same time the parsing is speeded up substantially which should not change with further development of the parser. The regular expressions can be written very easily and quickly.

6. Acknowledgements

This research is being supported by the Ministry of Education of the Czech Republic project No. LN00A063 (Center for Computational Linguistics). The data (PDT) would not be available unless the grant No. 405/96/K214, of the Grant Agency of the Czech Republic, supported the treebank design. I also want to thank the anonymous reviewers who pointed out some unclear points in this paper.

7. References

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

Michael Collins, Jan Hajič, Eric Brill, Lance Ramshaw, Christoph Tillmann (1999). A Statistical Parser of Czech. In: *Proceedings of ACL 1999*, pp. 505–512, College Park, Maryland.

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

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

Goran Nenadić, Duško Vitas (1998). Using Local Grammars for Agreement Modeling in Highly Inflective Languages. In: *Proceedings of TSD 1998,* pp. 91–96, Brno, Czechia.

Goran Nenadić (2000). Local Grammars and Parsing Coordination of Nouns in Serbo-Croatian. In: *Proceedings of TSD 2000,* pp. 57–62, Springer LNAI 1902, Brno, Czechia.

Vladimír Petkevič (1999). Czech Translation of G. Orwell's '1984': Morphology and Syntactic Patterns in the Corpus. In: *Proceedings of TSD 2000,* pp. 77–82, Mariánské Lázně, Czechia.

Larry Wall, Tom Christiansen, Randal Schwartz (1996). *Programming Perl.* http://www.perl.org/.

Daniel Zeman (1998). *A Statistical Approach to Parsing of Czech.* In: Prague Bulletin of Mathematical Linguistics, vol. 69, pp. 29–37. Univerzita Karlova, Praha, Czechia.

Daniel Zeman (2001). *Parsing with Regular Expressions: A Minute to Learn, A Lifetime to Master.* In: Prague Bulletin of Mathematical Linguistics, vol. 75. Univerzita Karlova, Praha,Czechia.