Thin Parsing: A Balance between Wide Scale Parsing and Chunking

Jon Patrick and Pham Hong Nguyen

Sydney Language Technology Research Group

School of Information Technologies

University of Sydney

{jonpat,pham}@it.usyd.edu.au

Abstract

This work presents a type of parser that takes the process of chunking to the stage of producing full parse trees. This type of parser, denoted Thin Parsers (TP) in this work has the characteristics of: following a given grammar, creating full parse trees, producing only a limited number of full parse trees, parsing in linear time of sentence length. Performance standards on the Penn Tree Bank show results slightly under that of stochastic parsers but faster performance. Various types of Thin Parsers are presented.

1 Introduction

Bottom-up and Top-down methods of parsing, invented in 1950s, can parse in time which is an exponential function of the length n of an input sentence O(c^n). Tabular parsing methods such as the CYK (Cocke-Younger-Kasami) and Earley can parse context free languages in polynomial time O(n^3), a significant improvement compared with the previous parsers. These methods are usually called “traditional methods”. However these methods are relatively slow, as they produce the complete parse trees. Also they use exhaustive search to find all possible parse trees, subsequently leading to the problem of needing to select an optimal parse from a very large set of solutions. Hence, researchers have tried to find other parsing methods and/or other formalisms for natural languages. For example, some researchers have tried to convert natural languages into regular languages to parse them by finite state automata (Black 1989, Pereira 2003). Some others work with parsers based on LL or LR grammars/parsers. However, they still have been unable to gain anything but small improvements.

Recently, with the creation of large annotated corpora such as the Penn Tree Bank, probabilistic parsing methods based on their manifest grammars appear to be a useful solution for the parsing problem. In this situation, the induction of probabilistic context free grammars (PCFG) derived from the annotated corpora has become a new objective and many researchers have quickly developed improved performance (Ratnaparkhi 1994, Collins 1996).

A different use of tree bank corpora is made by the chunking or shallow parsing process (Abney 1991, Brants 1999, Tjong Kim Sang 2000).The main idea behind this parsing strategy is that both human and machine usually do not require full information of the parsing processes. These parsers usually produce only partial parse trees instead of full trees, which can be a disadvantage, but, the trade off is that the parse time becomes linear with the length of input sentences O(n). These parsers are called shallow parsers and the inference of probabilistic grammar rule sets is one of their useful characteristics.

Unfortunately, shallow parsers cannot be used in applications in which full parse trees are required. Under these circumstances there is a need to use traditional parsers or improve shallow parsers to produce full parse trees. The challenge here is to solve the problems created by inferring a parser from a corpus. The strategy used in this research is to give attention to improving the parser without recourse to comprehensive generative probabilistic models of the tree bank but rather by developing fast

Proceedings of the Australasian Language Technology Workshop 2004
Macquarie University, Sydney, December 8th, 2004. Copyright, Australian Speech Science & Technology Association Inc.

Accepted after full paper review

algorithms with accuracies rivaling that of probabilistic models.

Justification for giving dominant attention to speed performance in the context of an inductive inference strategy is well supported by some established heuristics and by certain utilitarian values. As well, a search of the literature has failed to reveal any substantial attempt to analyse, on the basis of computability and speed performance, the advantages of an inductive inference approach to constructing a language parser from tree banks.

Well recognized heuristics that have been established and support the motivation for this work are that language is predictable. It is well recognized that there is significant redundancy in language at the lexicogrammar and semantic levels and that corpus analysis has demonstrated there are reasonable predictors for subsequent phenomena from a given point in a sentence.

Furthermore, Memory Based Learning has established the principle that "forgetting is harmful" to performance (Daelemans, van den Bosch & Zavrel 1999) and so remembering is helpful.

Functional motivations include: large scale grammars can be better utilized by developing parser from their outputs; applications exist that don't need wide coverage grammars but rather high speed. Current research in speech recognition is moving into a phase of exploiting lexicogrammar to improve performance (Lemon, 2004).

This paper promotes and develops a specific class of parsing methods, Thin Parsers (TP). Like traditional parsers, this method considers natural languages as context free languages and/or probabilistic context free languages. The aims of Thin Parsers are to overcome some weak points of both traditional (full) and shallow parsers. In comparison to full parsers, the Thin Parsers should work faster in practice and produce one or a few parse trees instead of large number. The main advantage of TPs over shallow parsers is that they can build full parse trees instead of partial trees, however, thin parsers do not have the ability to infer new grammar rules as shallow parsers do. Thus Thin Parsers could be considered closer to full parsers than shallow parsers. The Thin parser can be considered as a balance between the issues of the language class, parser speed and accuracy, and the need for full parse information. Hence, in this research the focus is on building performance models which are potentially useful in real situations rather than to build competence models.

2 Pre-processing of the PTB

The TPs in this research are developed from the Penn Tree Bank (PTB). As with other corpora, the PTB has some errors. Some of them can be corrected automatically by program whereas the others require manual reviews and corrections.

As this research deals with CFGs, which lack the ability to smooth errors, better data for establishing a "gold standard" for both training and testing models is needed. From many authors working with the PTB, it appears that only Bod's (1995) research is based on cleaned data as is this study.

Each sentence of the PTB corpus is presented in three different formats: raw sentences, sentences with part-of-speech annotation, and skeletal syntactic bracketed sentences each in a separate file[1]. Correct matching of these triples across the three files is needed to build up the full parse trees. However, due to errors in the data, the numbers of sentences are quite different in each file, making the work of extracting triples much harder and requiring human intervention.

From the development of suitable software, around 22,000 triples or only half of the data were extracted automatically. After correcting the data manually, this has been increased to 34,000 triples. This data set is denoted as the "Large set". From these triples, a subset of 6,700 triples was constructed, denoted the "Clean set" by deleting any triples having null elements and/or having incorrect parse trees (the trees that have none or more than one *S* node at the top of tree). This set is used in this research. The new set is quite small, compared with the original data, but still enough for experimentation as it is much larger than the sets

[1] Raw sentences are not used in this work.

used in other research of a similar nature, for example Bod (1995) only used 500 sentences.

3 Finite State Automata (FSA)

The starting point for our modeling of the PTB is the inductive inference of FSAs for both the POS tag sequences and the phrasal tag sequences (see Fig. 1). Two tests using POS and Phrasal tags were run using 2 data sets (Large and Clean) to create a baseline for further work (Table 1). The results are quite similar with the Sekine's (1997) although they are slightly worse. The reasons appear to be: i) a smaller training set is used here (30,600 sentences compared with 47,219 sentences); ii) The training and testing here is on different sets.

The results for Phrasal tags are significantly better than POS tags (19.19 % compared with 4.22%). The main reasons are: i) the average length of sentences (14.71) is shorter (21.23), creating the potential for repeated structures to be more frequent; ii) the annotation set for POS (36 POS tags + 12 other tags) is much larger than in Phrasals (14 syntactic tags + 4 null elements) (Marcus 1993). These results are used as a base line for our research. However, they are still far from that of a good probabilistic parser (with over 70%).

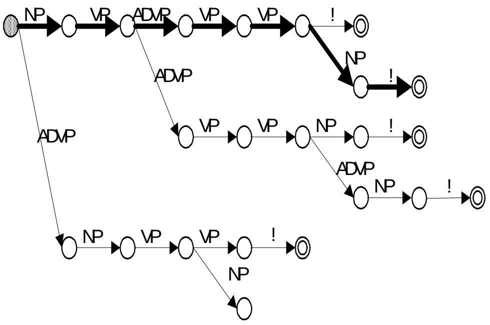

Figure 1. Inferred FSA for phrasal tag sequences in the PTB.

The Clean data set obviously brings some improvements. In the case of phrasal tags the FSA can parse successfully more than 44% of the phrasal test sentences. One of the reasons is that the average length of sentences is shorter. The POS tag results worsen between the Large and the Clean data sets compared to the phrasal tags. From these tests, it is evident that the lengths of sentences can affect the recognition results, the shorter the sentences, the better the result.

	POS	Phrasal
Large	4.22%	19.19%
Clean	3.66%	44.87%

Table 1. Coverage attained by inferred FSAs of PTB data sets.

In Charniak's (1996) work he ignored all sentences of length greater than 40 in the test data to avoid high parsing costs. An examination of the Big data set confirms his assessment that only a small sample are excluded. There are 29,462 sentences out of the 30,600 training set (96.28%) with a length of 40 words or under.

Our tests to identify relationships between the tag length and coverage show for POS tag sequences the FSA parser begins to show deterioration for lengths higher than 11 tags and for Phrasal tags about 14 tags.

3.1 Combination of POS and Phrasal FSAs

The POS tag FSA is useful in practice however, the main limitation is the low coverage. In contrast, the Phrasal tag FSA is not of much use even though it has a much higher coverage. To overcome the disadvantage of the first FSA and take the advantage of the second, they are merged into a new parser that works with inputs of POS and achieves a better coverage.

This new model can be viewed as two components, one is a simple FSA for recognizing phrasal tag sequences. The other part is a converter to convert sequences of part-of-speech tags into sequences of phrasal tags for input into the FSA. The Converter creates the mapping of the POS tags to phrasal tags. The Phrasal FSA after recognizing a string of phrasal tags identifies the tree top of a legal tree top in the training set to complete the full parse tree (see Fig 2).

3.1.1 Phrasal rules

The rule sets for converting POS tags into phrasal tags are extracted from the PTB. For example, from the tree in Figure 3, some phrasal rules as follows can be extracted:

Accepted after full paper review

NP→DT JJ NN NNS; VP→VBP; NP→NN; ADVP→RB; VP→VBG; QP→RB IN CD

From the 6700 sentences in the Clean dataset around 1200 phrasal rules were automatically extracted. This rule set is only a small subset of the PTB grammar, which has at least 10,000 rules as mentioned in Charniak (1996).

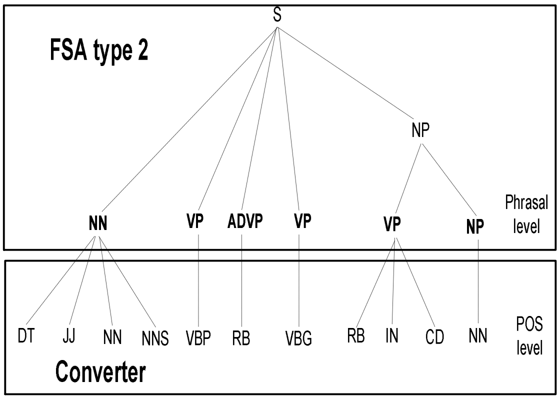

Figure 2. Combination of POS and Phrasal tag FSAs.

3.1.2 Limitations on a Converter

For parsing an unseen sentence a Converter is needed to perform an exhaustive search algorithm. That is it has to try every combination of POS tags to phrasal tag conversion for the set of POS tags in the sentence until the string of phrasal tags is recognized by the main FSA. However, this approach has exponential explosion even for a small rule set and short sentence.

3.1.3 Recursive Transition Network Induction – model TP1

In the first attempt to create an efficient Converter, the two kinds of FSA are combined by using Recursive Transition Networks (RTN) (Woods, 1970) because this model can help reduce the exponential time problem.

The two kinds of FSA are, one for part of speech tags and one for phrasal tags. The second one is the same FSA used in the initial tests above. The first one (type 1) uses a string of tags taken from the right side of phrasal rules instead of POS sentences. All rules which have the same left side are used to build one FSA. Thus there are many different FSA corresponding to a phrasal tag.

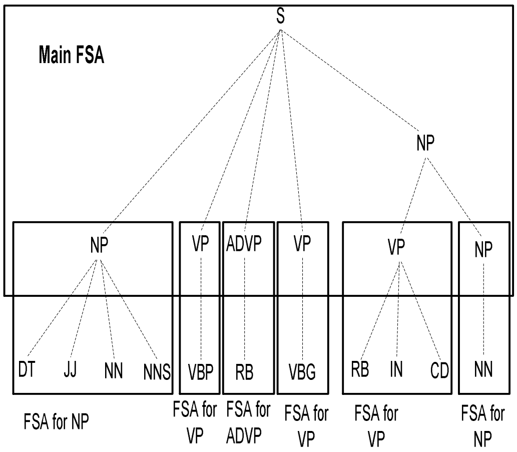

Figure 3. Combination of two types of FSA into an RTN.

3.2 Results and Analysis

The results of parsing by an RTN are shown in Table 2. The 6700 sentences of the Clean data set were used with 10 fold cross-validation. There is no restriction on sentence length for the first test, however, in the second test, like Charniak (1996), all the sentences with length in excess of 40 tags were removed. The RTN method has a very high coverage (98.32%) but the accuracy (R= 19%, P=23%, F=21%) are below the base line.

Theoretically, parsing by an RTN method is not as fast as using an FSA. Woods (1970) has shown that the time is about O(c^n). However, in this work the RTN performs much faster, the average parse time per sentence is 0.0176 second[2] and slightly better 0.0156 if the sentences which are longer than 40 tags are ignored because:

- the phrasal FSA has only a depth of one
- there are no recursive loops
- the first level is recognized as a chunk, not only a tag, thus, the number of elements at the first level is reduced significantly

[2] All tests are done on a computer P4, 1.6 Ghz, 1 GB Ram. The time includes time for parsing, extracting parse trees and measuring accuracy.

- the number of phrasal rules is around 1400 – much smaller than the total number of grammar rules of the PTB[3]
- this model stops parsing after finding a solution.

In the worst case, the maximum parse time for all data is 1.603 seconds and for <41 data is only 0.911 seconds[4].

The results in Table 2 of the RTN show a very high proportion of coverage – 98%. The high coverage and fast parse time of this model could well be useful for some applications.

N	**C'age**	**R**	**P**	**F**	**Time**
all	98.32	19.74	23.92	21.83	0.0176
<41	**98.48**	**19.18**	**23.3**	**21.24**	**0.0156**

Table 2. Performance percentages of RTNs on the Clean data (10-fold cross-validation).

4 Shallow Parsers for building Parse Trees

To establish an understanding of the base performance of shallow parsers when used to build full parse trees we developed a method based on Abney (1991). The first experiment showed the coverage of the system is only 0.5% (only 3 of 600 test sentences).

With the aim of producing a better assessment of coverage, the accuracy of the base sub-trees (lower parts of trees) which are created by the shallow parser is computed by an alternative method, that is the tag *S* is added to every incomplete parse tree. This produced the results of P=36.65%, R=15.03%. These results can be considered as another baseline. Comparisons of these results to the works of other authors are not possible as they have only reported the accuracy of chunkers.

5 Constructing Thin Parsers from Shallow Parsers – TP2

In this section, some improvements to the shallow parser model to extract the full parse trees are developed. These improvements can also be viewed as mixing models, that is a shallow parser and a finite state automaton (FSA), a shallow parser and a data-oriented parser (DOP), and the adaptation of a shallow parser with some non-deterministic components. The framework objective behind this blending is to reduce the number of levels needed in parse trees, reduce the number of elements of the parse trees, use of more stored pre-computed information, and to use non-deterministic mechanisms.

The large number of levels of the parse tree for the shallow parser is one of the major reasons for the poor coverage in extracting full parse trees. With each new level, the parser has more chance of being trapped and/or creating errors which will propagate to higher levels, that is the coverage and accuracy are affected seriously by the number of levels. Hence, the idea of improvement here is to let the parser work for the first level or at the most the few first levels only, which can work well. Then another process adds the top parts of trees to complete the full trees.

The tree tops that will be used to complete the parsing are pre-computed from the training corpus and saved into a database. One extra advantage of this method is that even though at the lower levels the trees are variable and large in number, at the higher levels the multiple forms decrease sharply. This means the number of tree tops is small, and easy to collect and to save to a database. Hence in testing the system has two parts. The first part works like a shallow parser and the second part is a search mechanism retrieving from this database.

After creating the base sub-trees of a parse tree (by working for only one or the first few levels), the shallow parser passes the result to the next process "Find and Join". This process retrieves from a database a saved tree top matching the tag configuration of these base sub-trees. If a suitable tree top is found, this function will attach it to the lower sub-trees to create a full parse tree. The condition to match the two parts

[3] An Earley parser (with parse time O(n^3)) was built and tested using a reduced set of 9000 rules of the PTB, in general this parser takes from 30 seconds / short sentence to some days for long ones (around 30 words). When using a much smaller and manually compacted rule set of 3000 the parser still takes several hours for a long sentence.

[4] The parse time is computed here to demonstrate how fast the model TP1 is in practice but not the fastest speed of this model.

is straightforward: the top sequence of tags of the base sub-trees must be matched with the bottom sequence of the tree top.

Although four methods can be considered for assessment (Pham, 2004), the test results are shown here with a weaker measure, adjusted crossover brackets, to enable comparison with other authors. The data for training and testing is the Clean data set (6000 sentences for training and 600 for testing). After chunking *k* levels (column *Level/k*) the system gets the results sequence which is recognized by an FSA. Column *Succ* shows the numbers of parsed sentences.

Model TP2 can parse successfully a range from 21.67% to 55.83% of input sentences with accuracies (*F*) from 76.61% to 91.68% (Table 3). The accuracies are highest when the shallow parser works for only the first level (level 0). The coverage, unlike accuracy, increases by level and reaches the highest proportion at level 7 (55.83%) before decreasing slightly. As previously discussed, with the higher levels the variety of tree tops decreases sharply, helping the system match these tops more successfully. The model TP2 can parse successfully over 46% of total sentences with high accuracy of over 76% at k>=5.

Level/ k	**Suc c.**	**Cover age %**	**P %**	**R %**	**F %**
1	239	39.8	92.3	91.1	91.7
2	130	21.7	90.4	89.7	90.0
3	219	36.5	82.3	83.1	82.7
4	248	41.3	79.9	80.5	80.2
5	277	46.2	78.5	79.3	78.9
6	304	50.7	78.8	79.4	79.1
7	335	55.8	77.3	78.1	77.7
8	307	51.2	76.5	76.7	76.6
9	316	52.7	76.6	77.1	76.8

Table 3. Results of model TP2 chunking, measures of the accuracy of nodes with adjusting crossover brackets.

6 Reduction of the number of tree elements - TP3

The number of grammar rules available for use in a parse tree is usually large. For each extra rule that is used in a parse the tree has more variation at higher levels. This provides more opportunities for the parse to be trapped or to propagate more errors. Furthermore, if two rules have the same right hand side, then the one with the highest frequency is used for building the tree. However this selection process of removing a less frequent rule is the main reason for shallow parsers not being able to build up a full parse.

Another idea for improvement of model TP2 is to reduce the number of grammar rules per parse tree by using the tree fragments which could be larger than these rules. By such a mechanism, the parser has more chance to use maximally frequent fragments, that is combinations of grammar rules, rather than grammar rules alone, thus avoiding the omission of less frequent grammar rules. Another advantage is that the number of levels can be reduced and so gain the benefit of getting higher coverage as well as faster execution time.

This idea is partly similar to Bod’s (1995) model of DOP for using tree fragments, however in detail, there are differences in the ways tree fragments are collected and the method of parsing.

6.1 Collection of fragments

In the DOP model, Bod collects all possible tree fragments. However, this has the disadvantage that the number of fragments grows exponentially with size of grammar.

In the case of the TP2 model, tree fragments are collected based on the data set prepared for the shallow parsing experiments, that is the filled trees, that is all branches have nodes at each lower level. The conditions for collecting a sub-tree as a real fragment are that all leaves at the end of the tree start on the same level, as the tree is filled, and, it has a height of 2 levels or more.

For example, Figure 4 shows a sub-tree of a parse tree. The DOP model will collect a total of 32 fragments from this sub tree. In the TP3 model, firstly this sub tree is converted into a filled tree as shown in Figure 5.a. From this data only two fragments are collected as in Figures 5.b and 5.c, hence an exponential increase of fragments is prevented.

6.2 Results

All data for training and testing model TP3 is reported for both shallow parsing (chunking, Table 4) and parsing (Table 5).

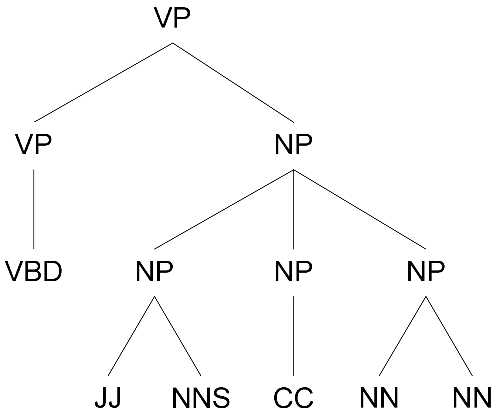

Figure 4. An example of a sub-tree from a parse tree.

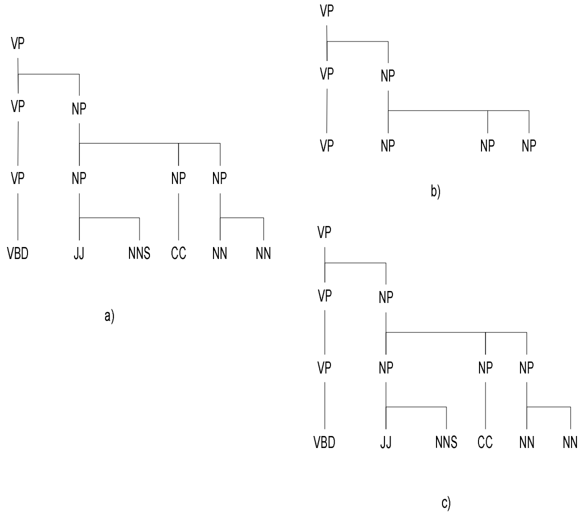

Figure 5. The parse sub-tree from Fig. 4 converted into filled sub-tree (a). From (a) in training only two fragments (b) and (c) are collected in model TP3.

Table 4 shows the results of chunking the Clean data set. This is quite similar to Table 5.3 text chunking except: the maximum level is only 1, and, the accuracy of the two highest levels appears to be exceptional because the number of sentences is quite small (844 and 586 respectively), compared with whole learning set (6000 sentence).

As the levels increase the number of sentences available for both training and testing decreases as there is a variable range of tree heights across the corpus.

Level	Precision	Recall	F
0	84.2	83.7	83.9
1	72.5	64.1	68.1
2	75.4	59.0	66.2
3	68.8	53.1	59.9
4	68.1	52.8	59.5
5	72.1	65.4	68.6
6	79.9	68.3	73.6
7	91.1	85.4	88.2
≥8	100.0	100.0	100.0

Table 4. Text chunking results using the Clean data set for model TP3.

Table 5 shows the results of the parse of 600 sentences from the Clean data set. The coverage of TP3 in the best case is over 61% (at level 5), better than the best coverage 55% of TP2. However, the accuracies in general are a little worse than the accuracies of TP2. Both TP2 and TP3 reach the highest accuracies at level 1 and reduce at higher levels.

Level	Succ.	Coverage	P	R	F
1	230	38.4	84.8	85.5	85.2
2	135	22.5	64.8	73.7	69.2
3	184	30.7	63.8	74.7	69.3
4	351	58.6	64.7	74.8	69.8
5	368	61.4	69.3	76.8	73.1
6	183	30.6	70.7	79.2	75.00
7	62	10.4	68.7	75.7	72.2
≥8	1	0.17	73.5	83.3	78.4

Table 5. Parsing results using the Clean data set with model TP3.

7 Non-Deterministic models –TP4, TP5

Two approaches have been investigated in an attempt to produce concurrent solutions to improving coverage and accuracy. Whilst the methods will be reported elsewhere the results are relevant to this study and show great promise. The first method, model TP4, uses the idea of a non-deterministic chunker with a deterministic attacher. The second model, TP5, uses a deterministic chunker with a non-deterministic attacher. By exploitation of the learning process of creating a tree tops database and using algorithms to match the database entries against the candidate parse trees rather than the other way the practical implementation requires linear time to parse. The performance of the two systems is presented in Table 6.

Model	Coverage	P	R	F
TP4	63.7%	83.0	84.6	83.8
TP5	82.2%	88.3	88.0	88.1

Table 6. Performance statistics of models TP4 & TP5.

8 Conclusions

Some improvements to building a thin parser from a shallow parser have been presented in the form of models TP2 and TP3, TP4 and TP5. They all include in the first stage a shallow parser. In general, these models can solve to some degree some problems of shallow parsing when extracting full parse trees. The problems are: deterministic mechanisms are not adequate enough for dealing with the ambiguities of natural language; the configuration of connections between layers make any error that occurs in the lower levels propagate to higher levels and worsen the performance; the Chunker and Attacher need to work with some limitations, such as using only the most probable rules; and not attach chunks with a size of 1. The results of these models are quite good: the coverage (from 0.5% of original shallow parser) has improved to more than 82%, and the accuracy has improved from 20% to over 88%.

9 References

Abney S. 1991. Parsing By Chunks. In: R. Berwick, S. Abney & C. Tenny (eds.), *Principle-Based Parsing.* Kluwer Academic Publishers, Dordrecht.

Black, A. (1989). Finite State Machines from Feature Grammars, *International Workshop on Parsing Technologies*, Carnegie-Mellon University, Pittsburgh, PA.

Bod R. 1995. Enriching Linguistics with Statistics: Performance Models of Natural Language, *Academische Pers*, The Netherlands. 143 pp. (Ph.D. thesis).

Brants T. 1999. Cascaded Markov Models. In *Proc of 9th Conf. of the Euro. Chapt of the ACL, EACL-99*, Norway.

Charniak E. 1996. Tree-bank grammars, *Tech Report CS-96-02*, Dept of Computer Science, Brown University.

Collins M. 1996. A New Statistical Parser Based on Bigram Lexical Dependencies. Proc of the *34th Annual Meeting of the ACL*, Santa Cruz.

Daelemans, W, van den Bosch, A & Zavrel, J. 1999. Forgetting exceptions is harmful in language learning. *Machine Learning*

Lemon, O. 2004. Context sensitive speech recognition in ISU dialogue systems: Results for the grammar switching approach. *Proc. 8th Workshop on the Semantics and Pragmatics of Dialogue, CATALOG'04.*

Marcus M., Santorini, B. Marcinkiewicz, M. 1993. Building a large annotated corpus of English: the Penn Treebank, *Jnl of Computational Linguistics*, Vol. 19 No.1.

Pham Hong Nguyen 2004. Thin Parsing, PhD Thesis, Univeristy of Sydney.

Tjong Kim Sang E. F. and Buchholz, S. 2000. Introduction to the CoNLL-2000 Shared Task: Chunking. In *Proc of CoNLL-2000*, Lisbon, Portugal.

Sekine S., Grishman R., 1995, A Corpus-based Probabilistic Grammar with Only Two Non-terminals. *Fourth International Workshop on Parsing Technology*; Prague.

Sekine S. 1998. Corpus-based Parsing and Sublanguage Studies, *Ph.D. Thesis*, New York University.

Sha F. Pereira F., 2003. Shallow parsing with conditional random fields. In *Proc of HLT-NAACL*. ACL.

Tjong Kim Sang, E. F. 2000. Text Chunking by System Combination. In *Proc of CoNLL-2000*, Lisbon, Portugal.

Ratnaparkhi A., Reynar J., and Roukos S., 1994. A Maximum Entropy Model for Prepositional Phrase Attachment. In Proc *of the ARPA Human Language Technology Workshop.*

Woods W. A. 1970. Transition Network Grammars for Natural Language Analysis, *Comm of the ACM,* 13, 10.