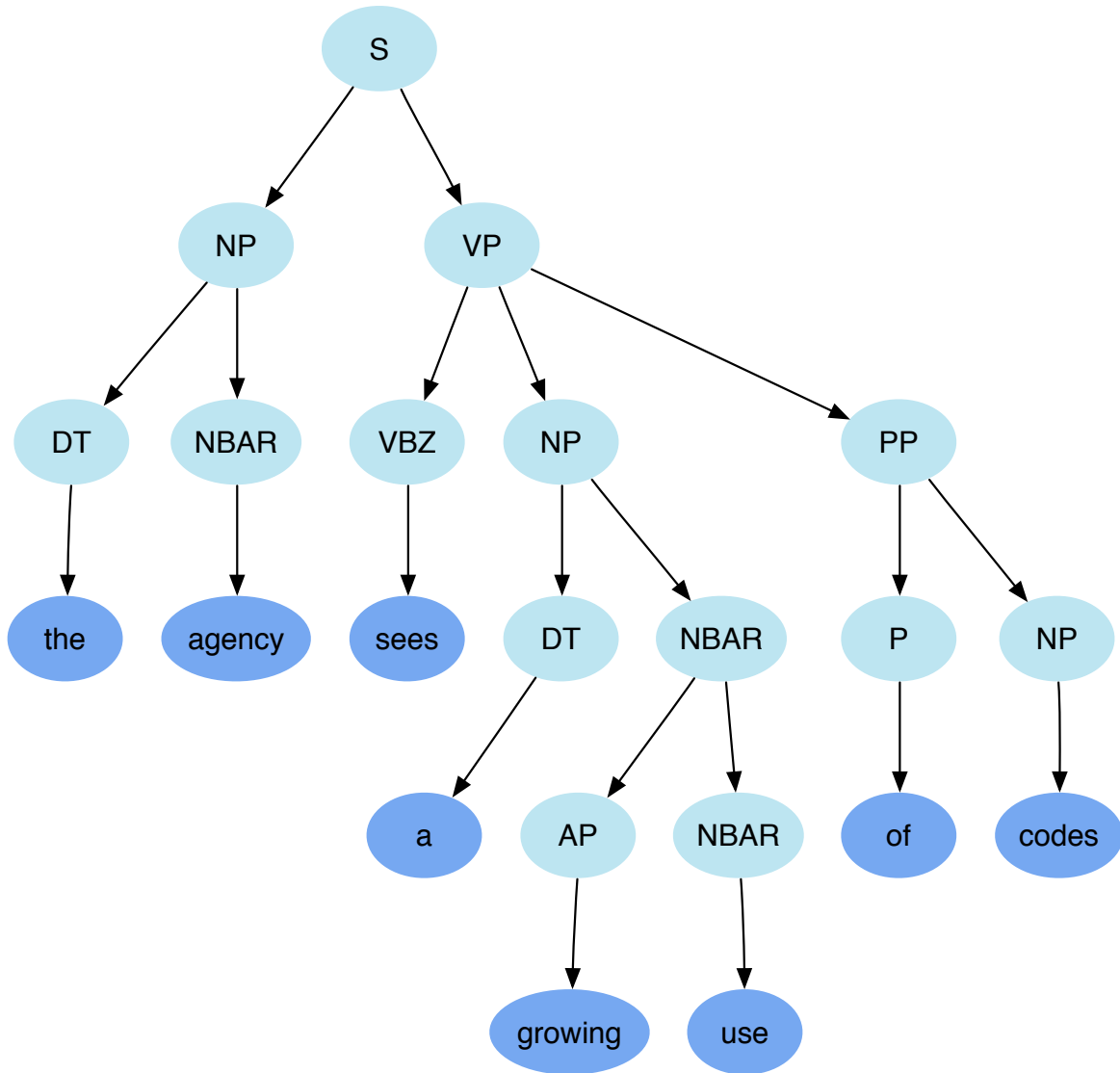


Language and Speech Processing

Probabilistic-Parser

Bart J. Buter - 9910522
Nicholas Piël - 0104612
Sanne Korzec - 0252700
{bjbuter,npiel,skorzec}@science.uva.nl



Graphically rendered output of the parser

1 Introduction

The construction of a sentence's structure is vital in understanding language. Without it a comprehension of dependencies between words or between sentence parts within a sentence is impossible to obtain.

In our previous paper [2] we discussed part-of-speech(POS) tagging, the task of labeling every word in the sentence with their POS-tag. That POS-tag model assumed a sequential syntactic structure. However, sentences seem to have a more complex non-sequential syntactical structure. Consider the following sentences:

- Where is the bus to the central station?
- Where is the bus from the dam to the central station?
- Where is the bus namely to the central station

The second sentence added "from the dam" to the first sentence, this sentence expands on the meaning of the first sentence, though the dependencies between to and station have not changed. The third sentence has substituted "from the dam" for "namely". This substitution leads to an ungrammatical sentence, though we can imagine other substitutions which would be grammatical.

The behavior, of non-sequential structure and substitutability as seen in the examples above can be captured when we model the syntactic sentence structure to be a tree. In such a tree we are allowed to substitute any subtree rooted at a nonterminal node by any other subtree as long as the "class" of the roots of both subtrees remain the same. It turns out that this behavior matches that of a context free grammar(CFG).

In this paper, we will describe a probabilistic parser, a program that produces the syntactic tree structure of a sentence. The algorithm uses a variant of the Cocke-Younger-Kasami(CYK) algorithm to generate the most probable or Viterbi-parse of a sentence given a probabilistic context free grammar(PCFG). We will report on research relevant to probabilistic parsers and PCFGs. Their foundational theory and implementational issues. We will research the performance of our parser and examine the influence of sentence length to the performance.

In section 2 we describe the theory which forms the basis of our parser. Section 3 describes the most important algorithms. Section 4 is used to describe our research methodology. Section 5 shows the results to our experiments performed with the parser and in section 6 we conclude our report.

2 Theoretical Model

In this section we will describe the theory which forms the basis for our probabilistic parser. We will describe CFGs, probabilistic grammars, Chomsky normal form(CNF), the CYK algorithm and related theories and concepts.

2.1 Languages and Grammars

2.1.1 Formal languages

In the Introduction we gave some intuitions into several properties of natural languages. Formal language theory provides us with models for natural languages. This is not to say that natural languages can be formally defined, however languages and their properties can be approximated, thus modeled by a formal representation.

A formal language L is the set of all well formed formulas(wff), a formal grammar G for this language recursively defines all wffs. The set of all wffs generated by a grammar G is denoted by $L(G)$. Given an utterance(sequence of words) u one can test if it is a wff for language $L(G)$ if according to grammar G we can generate u . This shows the two ways in which a grammar can be viewed, both as a generator of all wffs and as a recognizer of a wff.

This model resembles the noisy channel model, in which a sender sends a message M through a noisy channel and a receiver receives a message M' . It is the task of the receiver to make an estimate \hat{M} of the original message M , using the information obtained from M' . If we assume that all wff in a natural language are generated from a grammar G . Then the received messages M' are the wff(sentences) of the language and we can make an estimate \hat{G} of G .

2.1.2 Grammar preliminaries

A grammar can generate a wff through productions. These productions can be viewed as rewrite rules with a left-hand-side(lhs) and a right-hand-side(rhs). To generate a wff you start with the start-symbol, then successively apply the productions to rewrite the string until it only consists of terminal symbols. Productions describe that string $\gamma\alpha\delta$ can produce string $\gamma\beta\delta$, if there exists a production $\alpha \rightarrow \beta$.

Formally a **grammar** is a tuple $G = (N, \Sigma, R, S)$ where:

- N is a finite set of nonterminal
- Σ is a finite set of terminal symbols, symbols disjoint from N called the alphabet
- R is a finite set of productions of the form $\alpha \rightarrow \beta$, where $\exists x(x \in \alpha \wedge x \in N)$
- S is a unique member of N called the start symbol

We say that string η "**can produce**" string κ , denoted by $\eta \Rightarrow \kappa$ if there exists:

- a production $(\alpha \rightarrow \beta) \in R$
- strings γ and δ such that $\eta = \gamma\alpha\delta$ and $\kappa = \gamma\beta\delta$.

Using the " \Rightarrow " we can define a **derivation**, denoted $\phi_0 \Rightarrow \psi$ is a finite sequence of strings ϕ_1, \dots, ϕ_n such that:

- $\phi_0 \Rightarrow \phi_1 \Rightarrow \dots \Rightarrow \phi_n \Rightarrow \psi$.

The set of all derivations for ϕ will be denoted as $Der(\psi)$.

Given a grammar $G = (N, \Sigma, R, S)$, we define a **parse** T as a tree, where the root is S , the leaf nodes are in the set of terminal nodes Σ and the internal nodes of the tree are in the set of nonterminal nodes N . In a parse a nonterminal node and its children correspond with a production in R .

With $T(\alpha \Rightarrow \beta)$, we denote the parse associated with the derivation $\alpha \Rightarrow \beta$. A parse is sometimes defined as the left-most generation of an utterance. $G(u)$, we will denote the set of all parses that generate u .

2.1.3 (P)CFG

A CFG is a grammar with a restriction on the allowed productions. The only allowed productions are of the form:

- $\alpha \rightarrow \beta$, where $\alpha \in N$ is a single non-terminal symbol.

This means that the contexts of nonterminals are irrelevant when they are rewritten. This is an assumption that oversimplifies natural language, because humans do make use of context. This can be the context of the person we are talking to, the context of previous sentences etc.

To be able to calculate the most probable parse, we want to extend the CFG with probabilities.

A PCFG first proposed in [3] is a CFG with an additional probability mass function $P : R \rightarrow (0, 1]$, such that $\forall \alpha \in N : \sum_{\alpha \rightarrow \beta \in R} P(\beta|\alpha) = 1$. Thus every production has a supplementary probability.

2.1.4 Chomsky normal form(CNF)

A (P)CFG has an undesirable characteristic when we want to use it for computations. A production of the form $\alpha \rightarrow \beta$, has 1 nonterminal as lhs, namely α , but the rhs has no restrictions. This variety of rhs' makes it difficult to develop concise algorithms for CFGs, luckily there is the CNF.

The CNF is a CFG with the restriction that it only has productions of the forms:

- $\alpha \rightarrow \beta_l \beta_r$, where $\beta_l \in N$ and $\beta_r \in N$ are single non-terminal symbols.
- $\alpha \rightarrow \beta$, where $\beta \in \Sigma$ is a single terminal symbol.

A PCNF is defined similar to the PCFG is for the CFG.

2.1.5 Probabilities and parses

Given a parse $T(S \Rightarrow u)$, we want to compute the probability of the parse $P(T(S \Rightarrow u))$. We require that all derivations with the same parse have the same probability. In our probabilistic model this means that the order in which the productions are used should not influence the probability of a parse. Therefore, with $\phi_0 = S$ and $\phi_{n+1} = u$ we get:

$$\begin{aligned} P(T(S \Rightarrow u)) &= P(S \Rightarrow u) \\ &= P(S \Rightarrow \phi_1 \Rightarrow u) \\ &= P(\phi_0 \Rightarrow \phi_1 \Rightarrow \phi_{n+1}) \\ &= P(\phi_0 \Rightarrow \phi_1)P(\phi_1 \Rightarrow \phi_{n+1}) \\ &= \prod_{i=0}^n P(\phi_i \Rightarrow \phi_{i+1}) \end{aligned}$$

For CFGs we have made another independence assumption, namely that the productions are independent of the context. If we include this assumption in the probabilistic model we can simplify the formulas given above to:

$$\begin{aligned} P(S \Rightarrow u) &= \prod_{i=0}^n P(\phi_i \Rightarrow \phi_{i+1}) \\ &\approx \prod_{i=0}^n P(\gamma_i \alpha_i \delta_i \Rightarrow \gamma_i \beta_i \delta_i) \\ &\approx \prod_{i=0}^n P(\alpha_i \rightarrow \beta_i) \\ &\approx \prod_{i=0}^n P(\beta_i | \alpha_i) \end{aligned}$$

with $\phi_i = \gamma_i \alpha_i \delta_i$ and $\phi_{i+1} = \gamma_i \beta_i \delta_i$.

Thus we can calculate the probability of a parse by taking the product of the probabilities of all the productions used in the parse.

2.2 Problems with (Treebank)PCFGs

There are some concerns in using PCFGs for natural language processing. To name a few; The probability of a parse is obtained by a multiplication of values in the range $(0, 1]$, therefore there is a bias towards simpler parses. These short parses get a higher likelihood, because with the length of parses the number of multiplications increase, thus generally the lower the resulting probability.

Further natural languages have behaviors, such as long distance dependencies and selectional preferences which are not modeled by a PCFG.

- Did he park his car? vs. Did he park his just bought brand new car? shows a long distance relation.
- He biked to school vs. He biked to apple. shows a selectional preference of the verb biking for locations.

There are more objections that can be made against PCFGs for NLP however, results by Charniak on non-lexicalized treebank grammars[7], show surprisingly good performance without any smoothing on the Wall Street Journal corpus. Thus even though the PCFG is a simplified model of natural language, it still captures much of the syntactic structure of natural languages.

3 Algorithms

3.1 Translating PCFG to PCNF

If we develop a method to translate a PCFG to PCNF and back, then any algorithm that can be applied to a PCNF, can also be applied to the translated PCFG.

When we have a production $\alpha \rightarrow \beta$ in the PCFG we have 3 cases to translate:

- A singular rhs, where $\beta \in N$.

- A list rhs, where β is a list of more than 1 terminals or nonterminals, where β does not conform to the CNF restrictions.
- A CNF rhs, where the β does conform to the CNF restrictions, thus no translation will be needed.

A singular rhs needs to be recursively expanded, for example a grammar with production $S \rightarrow A, A \rightarrow BC$. The production $S \rightarrow A$ is not allowed by CNF, therefore it has to be expanded to $S \rightarrow BC$. The total probability assigned to A in the original rule should be preserved and given to its expansion, together with the relative probabilities of the new rhs' to each other. Thus when there is a singular rhs, we find all productions where this singular rhs is used as lhs, the newly found rhs' are used instead of the old rhs in the original rule. The recursion is needed because an expansion might again lead to a singular rhs. The recursion can cause expansions to come into an infinite loop. This infinite recursion will occur with rules of the form $S \rightarrow S$, or when these loops occur after multiple expansion steps. These infinite loops can be detected by keeping a list of previously expanded nodes, when a previously expanded node is again expanded we run into a loop. Some extra book-keeping is thus needed to conquer this problem, but it is not insurmountable.

Expansion may lead to orphaned productions these are productions with a non start lhs that is never used as rhs. After expansion of singular rules we can remove orphaned rules to keep our grammar minimal. This should be done recursively because removal of an orphaned production might lead to new orphans.

The expansion of singular productions and removal of orphaned productions is irreversible, because any reference to the expanded rhs are fully removed from the resulting CNF.

When we encounter a list rhs we need to augment our grammar with new nonterminals. Figure 1 show how a CFG G_{CFG} can be converted into a CNF G_{CNF} . The production $S \rightarrow 0A$ is not allowed in CNF. This is solved by augmenting the set of nonterminals with a nonterminal θ . In the original production the terminal 0 is replaced with θ , and a new rule is added $\theta \rightarrow 0$, such that θ always produces 0.

We also see the non CNF rules $S \rightarrow AAA$ and $S \rightarrow AAAA$. We have chosen to always split these rhs into the most left symbol and the remainder. The remainder is replaced by a nonterminal symbol which always produces the rhs. New productions that accomplish this are added to the grammar. It is noteworthy that the new nonterminal symbol aa produced in augmenting $S \rightarrow AAA$ can be reused for the augmentation of $S \rightarrow AAAA$. We mark the newly added rules so we can later recognize them, and fully invert the translation and transform back from CNF to CFG.

3.2 The CYK Algorithm

3.2.1 Simple CYK

Our main motivation to convert a CFG to a CNF was that we can then create a parse forest by applying the CYK algorithm [4, 5, 6]. A parse forest packs different parses of an

Figure 1: translation G_{CFG} to G_{CNF}

$$\begin{aligned} N_{CFG} &= \{S, A\}, \Sigma_{CFG} = \{0, 1\} \\ R_{CFG} &= \left\{ \begin{array}{l} S \rightarrow 0A|AAA|AAAA, \\ A \rightarrow 1 \end{array} \right\} \end{aligned}$$

$$\begin{aligned} N_{CNF} &= \{S, A, \theta, aa, aaa\}, \Sigma_{CNF} = \{0, 1\} \\ R_{CNF} &= \left\{ \begin{array}{l} S \rightarrow \theta A|Aaa|Aaaa, \\ A \rightarrow 1, \\ \theta \rightarrow 0, \\ aa \rightarrow AA, \\ aaa \rightarrow Aaa \end{array} \right\} \end{aligned}$$

utterance u with length n into a dense graph of polynomial size $O(n^2)$.

By making a small adjustment to the CYK algorithm we will not only find which trees are valid but also the most probable or Viterbi-parse for a given utterance.

The CYK algorithm is a dynamic programming method that creates a parse forest in a bottom-up approach in $O(n^3)$, see also algorithm 1. The algorithm starts with sub sequences of length 1 and checks if there is some kind of production rule $\alpha \rightarrow \beta$ where $\beta \in \Sigma$ is one of the terminal symbols. It will store the result of these production rules in the parse forest as Boolean value denoting a valid or invalid connection.

After this the algorithm continues with sub sequences of length 2, where it will consider every possible partition of the sub sequence into two parts where it checks if there is some production rule $\alpha \rightarrow \beta_l \beta_r$ where β_l should match the first part and β_r the second part and both should point to a valid connection. This procedure is repeated until the start symbol (TOP) has been reached. The utterance u is not a wff when the start symbol cannot be reached.

3.2.2 Extending CYK with Viterbi

With the described parse forest we only know the set of valid parse trees $G(u)$ under a grammar G given an utterance u , in order to select the most likely parse, we can extend the CYK algorithm with probabilities. In this case we will not only store whether a connection is valid but also the probability of the connection itself.

These probabilities are directly obtained from our PCFG. By storing the most probable partial parse for every lhs in every location of the parse forest with its probabilities we can obtain the most probable parse tree T :

$$\arg \max_{T \in G(u)} P(T|G).$$

Algorithm 1 The CYK algorithm

```
n ← length(U)
P[n,n,|G|] ← false.
FOR each i = 1 to n
  FOR each unit production  $\alpha \rightarrow \beta$ 
    P[i,1,j] ← true.
FOR each i = 2 to n
  FOR each j = 1 to n-i+1
    FOR each k = 1 to i-1
      FOR each production  $\alpha \rightarrow \beta\gamma$ 
        IF P[j,k, $\beta$ ] AND P[j+k,i-k, $\gamma$ ]
          P[j,i, $\alpha$ ] ← True

IF any of P[1,n,x] is True
  THEN RETURN True
ELSE RETURN FALSE
```

3.3 Estimating a PCFG

3.3.1 Treebank PCFG

Now that we have a formal knowledge of grammars and some useful algorithms, we want to train our PCFG from a treebank, the resulting PCFG is known as a treebank PCFG. A treebank tb contains sentences together with a representation of its parse, and the terminals and nonterminals are also known. We assume that all sentences are generated from some unknown CFG. First we add a TOP symbol to all parses to ensure we have a single start symbol from which all the given sentences can be generated. The representation of a parse in the treebank correspond to datastructures found in most high level languages, a simple conversion suffices to extract them.

From the treebank we can then obtain all productions together with their frequencies, denoted as $\Pi_{tb} = \langle R, F_R \rangle$, with R the set of productions and F_R the frequency of occurrence of this production in tb . From subsection 2.1.3 we know that production probabilities have the following constraints

- $\forall \alpha \in N : \sum_{\alpha \rightarrow \beta \in R} P(\beta|\alpha) = 1$, with $P : R \rightarrow (0, 1]$.

The relative frequencies of a production $\alpha \rightarrow \beta \in R$ is defined as:

$$rf(\alpha \rightarrow \beta, \Pi_{tb}) = \frac{F_R(\alpha \rightarrow \beta)}{\sum_{\alpha \rightarrow \beta' \in R} F_R(\alpha \rightarrow \beta')}$$

We will use this $rf(\alpha \rightarrow \beta, \Pi_{tb}) = \hat{P}(\alpha \rightarrow \beta|\alpha)$ as our maximum likelihood (ML) estimate of the probabilities for the productions.

4 Research Methodology

We implemented a probabilistic parser based on the models and principles described in section 2. We used a treebank with Dutch sentences about train traveling. The treebank

Figure 2: Distribution of sentence lengths in the testset

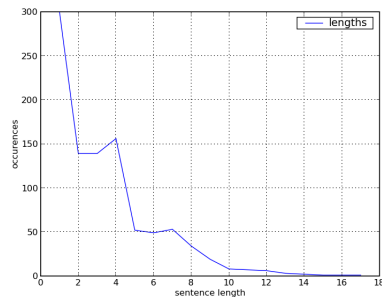


Table 1: Basic results

measure	%
exact match	88.61
coverage	96.80
exact match covered sentences	91.54
average precision	98.72
average recall	98.67

was divided in a training set with sentences and their parse, a test set with only sentences and a gold set with the true parses for the sentences in the test set. The test set included 1001 distinct sentences with an average length of about 3.4 words per sentence, while the trainingset had 9048 distinct sentences with a comparable average length. Thus both sets had a large distribution of small sentences, as can be seen in figure 2.

Research was done by measuring the exact match, coverage and PARSEVAL measures [1] for precision and accuracy of the parser on the test set. The PARSEVAL measures were also used to analyze the influence of the length of a sentence, in the results.

- Exact match, this is when the parser and the goldset give exactly the same parse.
- Coverage, percentage of test sentences that produce a valid parse.
- PARSEVAL measures also give partial credit to parses that do not have an exact match. Every nonterminal node in a parse produces a part of the sentence. This is used to produce a precision and recall measure.

The parser first generates a parse of the test sentence. This parse is then compared for exact matches, if the match is not exact, the PARSEVAL measures are calculated.

No attempt was made to smooth the model to be able to deal with unknown words, or ungrammatical sentences.

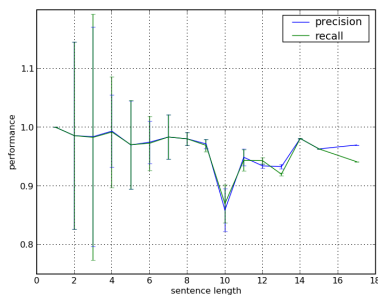
5 Results

5.1 Exact match and coverage

5.1.1 results

The results for exact match and coverage can be found in 1. The parser finds a parse for 96.80% of the sentences. Of the

Figure 3: Precision and recall vs. sentence length.



sentences that are covered 91.54% are parsed exactly as the goldset parse.

The parser can fail to cover a sentence in two ways. When an unknown word is in the test sentence the parser cannot find any productions that can generate this word, thus the word is not in the alphabet of the trained Grammar. This happens in 24 sentences.

When the parser cannot generate any derivation of the sentence, it will also fail. This means that according to the trained Grammar, the sentence is not a wff. This happens in 8 sentences.

5.1.2 Discussion

The coverage could be increased with smoothing techniques. These techniques could be employed to generate a production from a nonterminal to the unknown word. We might also extend the set of wff given the grammar, by abstracting productions. In natural languages a noun might be proceeded by a sequences of adjectives of any length. During training only a limited number of these sequences will occur in the trainingset. When we smooth the grammar in such a way that it allows a sequence of adjectives of any length as part of a wff, the grammar would cover more sentences and improve the coverage.

5.2 PARSEVAL

The average precision(98.72%) and recall(98.67) can also be found in table 1. These values are calculated over the covered sentences. The exact match of covered sentences(91.54%) is therefore the lower boundary for our precision and recall. Any partially correct parse in the sentences that are covered but not an exact match will increase the precision and recall.

5.2.1 Discussion

We know that Treebank PCFGs have a bias in favor of trees with less nodes (see subsection 2.2). This bias could explain the difference between precision and recall though the difference is too small to draw conclusions.

5.3 Sentence length

5.3.1 Results

Figure 3 shows the precision and recall, and error bars at 1 standard deviation, these error bars are a measure of the confidence of the data. Both precision and recall show almost the same results, both are generally slightly descending. The treebank consists of more short sentences, therefore the data on longer sentences is based on less observations. Short sentences show large standard deviations in their precision and recall. These variations are because the impact of a single wrong node in a parse has a larger influence on the final recall and precision, for shorter sentences.

5.3.2 Discussion

The sentence length does not have a big influence on the performance of the parser, because there is only a slight decline in performance for longer sentences. For some reason sentences of length 10 are harder to parser than others, we have not found any reason for this behavior.

6 Conclusion

From the results in table 1 and figure 3, it can be seen that a fairly simple probabilistic parser, can achieve a high performance. However it has to be noted that the domain of the treebank has been fairly strict and the sentences have on average only a length of 3.4. This being said, also on longer sentences there is no large drop in performance. These results are in line with results from Charniak [7].

References

- [1] Rehbein, Ines, van Genabith, Josef. 2007 Evaluating Evaluation Measures. In Proceedings of the 16th Nordic Conference of Computational Linguistics NODALIDA-2007.
- [2] Piël, Nicholas. et al. 2007. Language and Speech Processing. course report Universiteit van Amsterdam.
- [3] Booth, Taylor. Probabilistic Representation of Formal Languages FOCS 1969: 74-81.
- [4] Cocke, John, Schwartz, Jacob. 1970. Programming languages and their compilers: Preliminary notes. Technical report, New York University.
- [5] Kasami, T. 1965. An efficient recognition and syntax-analysis algorithm for context-free languages. Scientific report AFCRL-65-758, Air Force Cambridge Research Lab, Bedford, MA.
- [6] Younger, Daniel H. 1967. Recognition and parsing of context-free languages in time n^3 . Information and Control 10(2): 189-208.
- [7] Charniak, Eugene. 1996. Tree-bank grammars. In Proceedings of the National Conference on Artificial Intelligence (AAAI).