Chapter 6

Reducing structural ambiguity using a chunker

In chapter 4 an HMM was inferred from a corpus labeled with POS tags by the parser, and this HMM was subsequently used in a tagger to function as a POS tag filter at the early stages of parsing. Lexical ambiguity was thus reduced based on tagging preferences expressed by the parser itself, before full parsing took place. Looking at full parsing, an important type of ambiguity, in addition to the lexical type, is structural ambiguity; the parser has to decide which words are to be grouped together to form a plausible parse of the sentence. In a setup similar to the one used in chapter 4, the current chapter will present a method to reduce structural ambiguity based on a model inferred from parser-annotated data.

The technique used is that of chunking, which, in the context of language processing, is a form of shallow parsing. Techniques implementing shallow parsing aim at assigning low-level syntactic structure to an input sentence without attempting to combine those structures into phrases at a higher level as is done in full parsing. Depending on the task at hand, full parsing may return either too much or too little information [44]. In such cases, of which information retrieval, information extraction and tasks related to spoken language are typical examples, shallow parsing can provide exactly the information that is needed. Shallow parsing can also be used as a way of supporting a full-parsing system in order to reduce ambiguity and increase robustness.

In the approach described here, a statistical chunker is trained on data annotated by the parser with POS tags and chunk information. The chunker is then used to assign chunk structure to new input sentences, in the form of brackets that indicate where chunks begin and end. These bracketed sentences are used as input to the parser, and the parser will take the brackets
into account during parsing: the parser is forced to start and end syntactic phrases (of a type related to the type of chunk brackets used) at opening and closing brackets respectively. In this manner the number of possible parses is reduced.

The idea is motivated by the same technique already being used in interactive parsing using the Alpino parser, in the context of semi-automatic construction of a treebank [90]. In the interactive case, upon supplying the parser with an input sentence, the user can also add brackets in order to guide the parser in the direction of the particular analysis the user has in mind. While the interactive version allows for different kinds of brackets to be used, the automatic variant will only use brackets related to noun phrases.

It was mentioned in [66] that a variety of chunking methods did not outperform the advanced statistical parsers of [25] and [33]. However, it can be shown that the kind of structural information introduced above can help to make a parser more efficient. In table 6.1, parsing efficiency and accuracy are shown for parsing a set of sentences, first without brackets, then annotated with NP-related brackets as they would be used by the parser in what it believes to be the best analysis of these sentences. With these brackets available from the start there is an increase in efficiency, while accuracy does not drop. The full experiment is reported on in section 6.5.4, and the type of chunk to which this applies is introduced in section 6.5.1.

<table>
<thead>
<tr>
<th></th>
<th>parsing time (msec)</th>
<th>parsing accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>bracketed sentences</td>
<td>3338</td>
<td>87.60%</td>
</tr>
<tr>
<td>normal sentences</td>
<td>4449</td>
<td>87.52%</td>
</tr>
</tbody>
</table>

Table 6.1: Parsing efficiency (in average number of milliseconds needed per sentence) and parsing accuracy, for sentences with NP-related brackets previously assigned by the parser, and without brackets, respectively.

In the next section, the idea of chunking will be explained in more detail. In the rest of the chapter, different ways of implementing chunking using a HMM tagger are presented. One of the models used is the extended model presented in chapter 5. This time however, the extra context added to the model concerns chunks. It is tested whether the combination of POS tagging and chunking information in one and the same model has a positive effect on chunking accuracy. Analogous to how POS tagging was used to reduce lexical ambiguity for the parser, it will then be examined how chunk bracketing is used to reduce structural ambiguity for the parser.
6.1 Chunking

Chunking constitutes a shallow analysis of a sentence, grouping adjacent words into *chunks*. Chunks are defined as non-overlapping and non-recursive groups of words. After a chunking process has been applied to a sentence, not all words in the sentence are necessarily part of a chunk. A classification into different types of chunks can be made based on the types of syntactic phrases described by the grammar. For instance, the types of chunks that are often of interest are NP chunks and VP chunks. In chunking terminology, these non-recursive phrases are called baseNPs and baseVPs respectively. Of these two, only the baseNPs will be addressed in the work presented here. BaseNPs can be defined more precisely as non-recursive NPs from which post-modifiers are excluded. In example (1) baseNPs have been marked using brackets.

(1)  
[De eend] landde op [het dak] van [de boerderij]  
[The duck] landed on [the roof] of [the farmhouse]

Church [31] describes a way of marking simple noun phrases by inserting brackets between words based on the probabilities of inserting opening and closing brackets between the different parts of speech. Using these probabilities and given a sequence of POS tags, all possible non-recursive bracketings are considered and scored.

Abney [1] suggested that analyzing the structure of a sentence on the level of chunks can be useful as a first step in full parsing. The general idea is that the chunking step is applied as a means of reducing the workload for a more powerful form of parsing that is applied afterwards. The technique used for chunking is assumed to be relatively cheap compared to the full parsing machinery, but sufficiently accurate at its low level task to provide the next step in the process with reliable input. Abney describes an approach in which the two components in this setup are a chunker and an *attacher*. The chunker is applied first and finds the separate chunks, after which the attacher is used to combine chunks into sentences by attaching their parse tree fragments to each other.

6.2 Chunking as tagging

Up to this point chunking has been presented as the grouping of words by inserting opening and closing brackets in the sentence. Ramshaw and Marcus [72] formulated chunking as a tagging problem, where chunks are indicated by assigning chunk tags to the words themselves. In the simplest case these chunk tags indicate whether a word is inside or outside of a chunk. Ramshaw
and Marcus mention that the advantage of this approach is that it does not have to deal with ensuring that opening and closing brackets match. An additional advantage is that methods for POS tagging can now be used for chunking. In [88], four ways of representing chunks using tags are identified under the names IOB1, IOB2, IOE1 and IOE2. The chunk tags I, O, B and E indicate whether a word is inside of a chunk (I), outside of a chunk (O), at the beginning of a chunk (B), or at the end of a chunk (E), respectively. IOB1 is the method used by Ramshaw and Marcus. It uses a B to denote the beginning of a chunk only when this is necessary to separate the chunk from an adjacent chunk to its left. IOB2, as used for example in [73], always uses a B at the beginning of a chunk. IOE1 and IOE2 correspond to IOB1 and IOB2 respectively, but they explicitly mark the end of a chunk instead of the beginning.

<table>
<thead>
<tr>
<th></th>
<th>De jongen</th>
<th>gaf het meisje een bloem .</th>
</tr>
</thead>
<tbody>
<tr>
<td>IOB1</td>
<td>I</td>
<td>I O I I B I O</td>
</tr>
<tr>
<td>IOB2</td>
<td>B I O B I B I O</td>
<td></td>
</tr>
<tr>
<td>IOE1</td>
<td>I I O I E I I O</td>
<td></td>
</tr>
<tr>
<td>IOE2</td>
<td>I E O I E I E O</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.2: Example of a sentence to which chunk tags have been assigned according to the four different tagging schemes.

Table 6.2 shows these four tagging schemes being applied to a sentence. It can be argued that the IOB2 and IOE2 approaches are less ambiguous than the other methods as the same sequence of words will always receive the same sequence of chunk tags regardless of its context. In this work both the IOB1 and IOB2 tagging schemes will be used.

### 6.3 Different methods

Using a chunk tagging approach, the POS tagger introduced in chapter 4 can be used to label chunks. A number of techniques can be differentiated based on whether the chunks tags are assigned in a single step or following POS tag assignment, and on the nature of the chunk tags themselves. The four methods that are investigated in this chapter are called naive two-step, extended two-step, naive combined, and combined, and are summarized in table 6.3. In section 6.4 these methods will be compared in performing baseNP chunking.
6.3. Different methods

6.3.1 Naive two-step

This method entails two applications of the tagger. First the tagger is used with a normal POS tagging model on input sentences consisting of words. In the second step, the sequences of POS tags resulting from the first step are the input to the tagger, and chunk tags are assigned to the POS tags. An example is given in table 6.4, showing how the sentence in example (2) is tagged with baseNP chunk tags in two steps. The same example sentence will be used in the descriptions of the other methods.

(2) Controle op de naleving is een zaak van het rijk

‘Checking for compliance is a state matter’

\[
\begin{array}{cccccccc}
\text{Controle op de naleving} & \text{is} & \text{een zaak} & \text{van} & \text{het rijk} \\
\downarrow & & & & & & & \\
\text{tagger assigning POS tags to words} \\
\downarrow & & & & & & & \\
noun & prep & det & noun & verb & det & noun & prep & det & noun \\
\downarrow & & & & & & & \\
\text{tagger assigning chunk tags to POS tags} \\
\downarrow & & & & & & & \\
B & O & B & I & O & B & I & O & B & I
\end{array}
\]

Table 6.4: A chunking example using the naive two-step method; the boxes show tagger input and output.

6.3.2 Extended two-step

In the second step of the naive two-step method, chunk tags are assigned to POS tags. Typically the set of chunk tags used in tagging as chunking is small. As a result, the model used will contain information on only a small
Chapter 6. Reducing structural ambiguity

number of different chunk tag n-grams. It will be shown that better chunk
tagging results can be attained by extending the chunk tagset so that each
tag also contains the POS tag to which the chunk tag was assigned in the
training data. Thus, in the second step the POS tags are being tagged with
tags that are concatenations of POS tags and chunk tags. An example is
given in table 6.5. After tagging, the assigned tags are decomposed, leaving
just the chunk tag part.

| Controle op de naleving is een zaak ...
| tagger assigning POS tags to words
| noun prep det noun verb det noun ...
| tagger assigning concatenated chunk tags and POS tags to POS tags
| B_noun O_prep B_det I_noun O_verb B_det I_noun ...

Table 6.5: A chunking example using the extended two-step method.

6.3.3 Naive combined

As opposed to the above two methods, the next two will assign POS tags
and chunk tags in a single round of tagging. In the naive combined method,
this will be implemented by using a tagset that consists of concatenations of
chunk tags and POS tags. The combined tags are assigned to the words in the
input sentence, as shown in table 6.6, after which the tags are decomposed
so that only the chunk tag part remains.

| Controle op de naleving is een zaak ...
| tagger assigning concatenated chunk tags and POS tags to words
| B_noun O_prep B_det I_noun O_verb B_det I_noun ...

Table 6.6: A chunking example using the naive combined method.
6.3.4 Combined

In chapter 5 it was described how increasing the size of the tagset in the manner described for the naive combined method leads to an increase in the size of the HMM that can be avoided, to a certain extent, by adding the information to the states of the HMM separately from the tags. Since the different types of information are stored separately this allows for the definition of a model that is most appropriate for the task at hand, where relations between elements that are considered irrelevant are avoided. The combined method is an application of the same idea to chunking: the states in the extended HMM represent combinations of POS tags, chunk tags and observed words. Tagging using this HMM proceeds as normal, except that the extra chunk information is also used. In the final stage of tagging, the chunk tags instead of the POS tags are sorted on their probability, and the best chunk tag is selected. An example is provided in table 6.7. The bottom row of the table shows that separate POS tags and chunk tags are assigned to the input sentence.

```
Controle op de naleving is een zaak van het rijk
```

<table>
<thead>
<tr>
<th>noun</th>
<th>prep</th>
<th>det</th>
<th>noun</th>
<th>verb</th>
<th>det</th>
<th>noun</th>
<th>prep</th>
<th>det</th>
<th>noun</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>O</td>
<td>B</td>
<td>I</td>
<td>O</td>
<td>B</td>
<td>I</td>
<td>O</td>
<td>B</td>
<td>I</td>
</tr>
</tbody>
</table>

Table 6.7: A chunking example using the combined method.

The extended model

While the two-step methods and the naive combined method use the standard POS tagging model on a modified tagset, the combined method uses the extended model described in chapter 5. Chunk tags are added to the HMM for POS tagging as contextual information in the same way information was added to predict finiteness of verbs. That is, the same model is used, but the extra contextual information takes the form of chunk tags. Here this extended POS tagging model is repeated as equation 6.1, in which the $t_i$, $w_i$, and $c_i$ variables denote respectively the POS tag, word, and chunk tag at position $i$. 

```
Chapter 6. Reducing structural ambiguity

\[
\begin{align*}
P(t_i, w_i, c_i|t_{i-2}, t_{i-1}, c_{i-1}) & \approx \\
P(t_i|t_{i-2}, t_{i-1}, c_{i-1}) & \cdot \\
P(c_i|c_{i-1}, t_i) & \cdot \\
P(w_i|t_i)
\end{align*}
\]

(6.1)

In this model, the following simplifying assumptions are made: the current POS tag depends only on the two previous POS tags and the previous chunk tag, the current chunk tag depends only on the previous chunk tag and the current POS tag, and the current word only depends on the current POS tag. Different assumptions could be made that lead to other plausible models.

6.4 Combined tagging and chunking

The extended model described in section 6.3.4, used in the combined method, combines information about POS tags and chunk tags. In section 6.4.1 an experiment is conducted to see whether including the chunk information in the model decreases POS tagging accuracy.

In section 6.4.2 the four chunking methods are used to assign baseNP chunk tags. This experiment will be performed on a standard data set based on the Wall Street Journal corpus, in order to compare the results with previous work on baseNP recognition.

6.4.1 POS tagging

In chapter 5 it was shown how adding extra context to the model in a naive way increased its complexity in such a way that performance decreased. The proposed alternative method did not cause a decrease in POS tagging accuracy. An experiment is conducted to validate the expectation that extending the model in the same way with context information related to baseNPs does not decreases POS tagging accuracy, compared to the standard model.

Method and data

The extended model (displayed in equation 6.1) is inferred from training data using frequency counts. The tagger, using this model, is applied to a set of test sentences. Smoothing of n-gram probabilities and treatment of unknown words is performed as explained in chapter 4 (section 4.4.3) and chapter 5 (section 5.3.1) respectively. The input to the tagger consists of just the words; POS tags and chunk tags as assigned by Alpino are considered the correct taggings and are used to compute the accuracy of the tagger.
afterwards. In the case of this experiment, the accuracy attained on the
POS tags is computed. The standard tagging model is trained on the same
data (except that context information is not used) and applied to the same
test sentences. In both cases, the tagger assigns a single POS tag to each
word in the test sentences.

The data used spans the year 2002 of the Dutch daily newspaper *Parool*. However, in this experiment as well as in the experiments in the rest of the
chapter, only sentences of a length up to 20 words are used in training. Training data is 558,377 sentences (8.4 million words), test data is 500 sentences
(7750 words). The data consists of sentences labeled with POS tags and
chunk tags. The chunk tags are of the IOB2 type, described in section 6.2,
and concern only baseNPs. POS tags as well as chunk tags are assigned to
the newspaper text by the Alpino parser.

Results

The results are given in table 6.8. Using the relatively weak unigram model,
the chunk information increases POS tagging accuracy. As higher order mod-
els are used, this increase disappears, but no significant decrease in accuracy
is observed. It can be concluded that the added complexity caused by in-
cluding chunk information in the unigram, bigram, and trigram models does
not decrease their accuracy in modeling sequences of POS tags.

<table>
<thead>
<tr>
<th>model</th>
<th>standard</th>
<th>extended</th>
</tr>
</thead>
<tbody>
<tr>
<td>unigram accuracy</td>
<td>84.79%</td>
<td>86.21%</td>
</tr>
<tr>
<td>bigram accuracy</td>
<td>91.74%</td>
<td>91.78%</td>
</tr>
<tr>
<td>trigram accuracy</td>
<td>92.39%</td>
<td>92.32%</td>
</tr>
</tbody>
</table>

Table 6.8: POS tagging results using the standard model and the
model to which a separate chunk feature has been added as in
equation 6.1.

6.4.2 BaseNP chunking

In order to compare the described chunking methods to previous work,
baseNP chunking experiments are run using a standard data set made up
of Wall Street Journal (WSJ) data.
Method and data

In the combined method, tagging will proceed as normal, as described in section 6.4.1, using the extended model. However, what is extracted at the end is not the best POS tag for every word but the best chunk tag. As described in chapter 4, the tagger does not explicitly compute the most likely sequence of tags, but rather the most likely tag at each position in the sentence. The probability of an individual tag is computed as a combination of the forward and backward probabilities for the states in the trellis (a two-dimensional structure representing HMM states through time) that represent that tag. In chunking, instead of combining states that represent the same POS tag, states representing the same chunk tag are combined. Based on their forward-backward values, the chunk tags are ranked from most probable to least probable. In the experiment at hand, the single best chunk tag is selected.

The naive combined approach is similar to normal POS tagging, except that the tags are composed of POS tags and chunk tags.

In the two-step methods, as a first step POS tags are assigned to the words in the input sentence. In the second step, chunk tags are assigned based on the POS tags. Both steps can be performed by the tagger: in the second step, the tagger will use a modified POS tagging model in which chunk tags play the role of POS tags, and POS tags play the role of words. The extended two-step method differs from the naive two-step method in that the chunk tags used are composed of chunk tags (out of the IOB set) and POS tags. As with POS tagging, a lexicon stating which chunk tags (or extended chunk tags) were used for a given POS tag is constructed based on the training data. In the case of the naive two-step method, all chunk tags will be assigned if a POS tag is not in the lexicon; this is feasible in this particular approach because of the small size of the set of chunk tags. In the case of the extended two-step method, a set of tags associated with names will be assigned, as a safety measure to ensure that tagging can continue, if a POS tag is not in the lexicon; in practice these measures are never necessary as all POS tags will normally have been seen in the training data.

After chunk tags have been assigned, the chunks themselves can be straightforwardly derived by translating pairs of chunk tags into opening and closing brackets that designate the start and end of a chunk respectively. In assigning brackets, it is typically important to ensure that opening and closing brackets match. In the case of deriving brackets from IOB tags, this is not a problem: the tags indicate which sequences of words are chunks, and brackets can simply be placed around these sequences, always resulting in matching pairs. An example is provided in table 6.9. For instance, the OB tag se-
6.4. Combined tagging and chunking

A sequence assigned to the words *op de* (“on the”) is translated into an opening bracket between these two words; a B following an O indicates the start of a chunk. The tag sequence IO provides the corresponding closing bracket. (An overview of which pairs of tags lead to which brackets is given in table 6.13.)

| 1. | Controle op de naleving is een zaak van het rijk. |
| 2. | B O B I O B I O B I O |
| 3. | [ ] [ ] [ ] [ ] |
| 4. | [Controle] op [de naleving] is [een zaak] van [het rijk]. |

Table 6.9: Example showing how chunk brackets are assigned to a sentence through tagging with IOB tags.

The data used is the same data used by Ramshaw and Marcus [72] which has become a standard data set for the baseNP recognition task. The data consists of sections of the Wall Street Journal corpus: sections 15-18 are used for training (8,936 sentences, 212 thousand words) and section 20 is used for testing (2,012 sentences, 47 thousand words). The data contains the words, POS tags and IOB1 tags. The POS tags have been assigned using the Brill tagger [18]. This means that in performing the two-step approach to chunking, the first step, in which POS tags are assigned, has already been done.

Since the POS tags in the Ramshaw and Marcus data have been assigned using the Brill tagger, they are not necessarily correct. Although training the different models on the correct POS data would be more appropriate, the fact that in other research results have been attained on this particular data set is why this data is used here.

**Results**

Results are computed in terms of precision and recall attained in recognizing baseNP chunks. In addition, the F-score [97] is derived which combines precision and recall according to the following formula: $F = 2 \cdot \frac{precision \cdot recall}{recall + precision}$. The results are presented in table 6.10.

The baseline result at the bottom of table 6.10 is achieved by assigning to each POS tag the chunk tag that was most often associated with that POS tag in the training data. The naive two-step method performs only slightly better than the baseline. This low performance can be ascribed to the small set of chunk tags being used; in the second step of the naive two-step method, in which chunk tags are assigned to POS tags, the probabilities used by the tagger will be associated with just the small set of possible n-grams as constructed out of the tags I, O and B. This information is too
Chapter 6. Reducing structural ambiguity

<table>
<thead>
<tr>
<th>method</th>
<th>precision</th>
<th>recall</th>
<th>F-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>extended two-step</td>
<td>91.56%</td>
<td>91.17%</td>
<td>91.36%</td>
</tr>
<tr>
<td>naive combined</td>
<td>89.47%</td>
<td>89.81%</td>
<td>89.64%</td>
</tr>
<tr>
<td>combined</td>
<td>86.48%</td>
<td>86.06%</td>
<td>86.27%</td>
</tr>
<tr>
<td>naive two-step</td>
<td>79.11%</td>
<td>82.56%</td>
<td>80.80%</td>
</tr>
<tr>
<td>baseline</td>
<td>78.20%</td>
<td>81.87%</td>
<td>79.99%</td>
</tr>
</tbody>
</table>

Table 6.10: BaseNP recognition results on the WSJ data in terms of precision, recall and F-score, comparing the four approaches to chunking.

coarse to do precise tagging. If the chunks are extended with the POS tags to which they were assigned in the training data, more information is contained in the model and chunking results increase: the extended two-step method performed best of all four methods.

The combined method actually performs worse than the naive combined method. In chapter 5, the use of the contextual feature in the HMM was presented as a means of remembering a certain aspect of the clause being analyzed from one moment to the next; in chunking, the type of chunk tag to be assigned to a given position in the sentence may be more dependent on local context, and to a lesser extent a case of remembering sentence-wide information. The naive model does not assume a particular relation between the chunk tags, POS tags and words as much as the combined model.

Although the “chunking as tagging” approach allows for the use of a POS tagging HMM to do chunking, many other techniques have been tested on the same data. Ramshaw and Marcus [72] report an F-score of 92.0 using transformation-based learning. Kudo and Matsumoto [54] achieve an F-score of 94.22 using support vector machines, which is the best published result on this data. In work using system combination, where the output of different chunking systems is combined, F-scores of between 93 and 94 are reported [87, 86, 91].

Results on the IOB tags. Using the same data, the results were evaluated in terms of accuracy attained on the IOB chunk tags themselves, instead of on the chunks they imply. The results are in table 6.11 and reflect the same pattern as seen in the results measured on chunks. Ramshaw and Marcus, using transformation-based learning, report an IOB tag accuracy of 97.4%.  

Results on the IOB tags. Using the same data, the results were evaluated in terms of accuracy attained on the IOB chunk tags themselves, instead of on the chunks they imply. The results are in table 6.11 and reflect the same pattern as seen in the results measured on chunks. Ramshaw and Marcus, using transformation-based learning, report an IOB tag accuracy of 97.4%.  

Results on the IOB tags. Using the same data, the results were evaluated in terms of accuracy attained on the IOB chunk tags themselves, instead of on the chunks they imply. The results are in table 6.11 and reflect the same pattern as seen in the results measured on chunks. Ramshaw and Marcus, using transformation-based learning, report an IOB tag accuracy of 97.4%.
6.5. Reducing structural ambiguity

Table 6.11: Tagging accuracy for baseNP IOB tags on the WSJ data, comparing the four approaches to chunking.

<table>
<thead>
<tr>
<th>method</th>
<th>accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>extended two-step</td>
<td>97.28%</td>
</tr>
<tr>
<td>naive combined</td>
<td>96.58%</td>
</tr>
<tr>
<td>combined</td>
<td>95.67%</td>
</tr>
<tr>
<td>naive two-step (bigram model)</td>
<td>94.66%</td>
</tr>
<tr>
<td>baseline</td>
<td>94.48%</td>
</tr>
</tbody>
</table>

6.5 Reducing structural ambiguity

In chapter 4 an HMM trained on POS tagged data created by the parser was used to reduce lexical ambiguity in parsing new sentences. In a similar setup, the goal will now be to reduce structural ambiguity. The parser is used to assign both POS tags and noun phrase chunk structure to a large amount of text. From this labeled corpus, the model required by one of the chunking methods described in section 6.3 is inferred. Such a model can be seen as a finite-state approximation of the parser’s definition of noun phrase chunks. Next, the model is used in the tagger to assign chunk brackets to the input sentences using the selected method. In parsing a bracketed sentence, the parser is forced to start or end an NP at positions in the sentence that contain respectively an opening or closing bracket.

Although the tagger, using the IOB tagging scheme, indicates complete chunks and thus both their opening and closing brackets, in this setup the interest will be in individual brackets. The boundaries detected by the tagger will be used to restrict the possibilities of the parser, therefore a high level of bracketing precision needs to be ensured in order to avoid ruling out correct analyses. Since the tagger can assign different probabilities to the opening and closing bracket of one and the same chunk (as described in more detail in section 6.5.2), one may leave out one bracket independently of the other. The parser has been modified to deal with unmatching brackets in the input sentence.

6.5.1 Chunk data useful to the parser

Previous experiments concerned baseNPs. However, the parser does not work with baseNPs, but with normal NPs. Therefore the “chunks” considered in this application are complete NPs that do not contain other NPs. This type of NP will be referred to as innermost NP. An example sentence from section 6.1 has been repeated here as example (3), but this time only the
innermost NPs are marked.

(3) [De eend] landde op het dak van [de boerderij]
    [The duck] landed on the roof of [the farmhouse]

In this example, the phrase het dak ("the roof") is not marked as an innermost NP, since it is only the beginning of the complete NP. A test in assigning IOB2 tags for innermost NPs was run to see if using the different chunk type leads to different results compared to the baseNP experiments. Data in this experiment was the year 1999 of the Parool newspaper with POS tags and IOB tags for innermost NPs assigned by Alpino. The amount of training data is 574,917 sentences (9.0 million words), while test data is 500 sentences (8160 words).

The results are displayed in table 6.12 and are similar to those reported for baseNP tagging accuracy on WSJ data in section 6.4.2, as far as the relative performance of the different methods is concerned. In absolute terms the results on the innermost NP tags are lower, but since these are tests on different data sets this comparison is likely to be influenced by amounts of training data used and characteristics of the texts involved, in addition to the difference in chunk type. The baseline accuracy is considerably lower in this experiment, and the naive two-step method again offers limited improvement with respect to the baseline.

<table>
<thead>
<tr>
<th>method</th>
<th>accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>extended two-step</td>
<td>95.54%</td>
</tr>
<tr>
<td>naive combined</td>
<td>93.66%</td>
</tr>
<tr>
<td>combined</td>
<td>93.19%</td>
</tr>
<tr>
<td>naive two-step</td>
<td>84.01%</td>
</tr>
<tr>
<td>baseline</td>
<td>81.37%</td>
</tr>
</tbody>
</table>

Table 6.12: Tagging accuracy for innermost NP IOB tags on Parool data, comparing the four approaches to chunking.

### 6.5.2 Assigning brackets based on IOB2 tags

In previous experiments, the tagger was instructed to assign only a single chunk tag to every position in the sentence. Chunk brackets are then straightforwardly derived from the chunk tag sequences. However, this time the aim is to compute the probabilities of individual brackets, for which the probabilities of all chunk tags at the relevant positions in the sentence have to be taken into account.
In order to express the probabilities of brackets in terms of the probabilities of chunk tags, the fact that opening and closing brackets are associated with specific IOB tag pairs is used. The associations are listed in table 6.13. For example, in the first row of the table an O followed by a B is associated with an opening bracket between these two sentence positions. The event of not having a bracket between two adjacent positions is also associated with certain IOB tag pairs, and is represented in the table as the underscore symbol (\_). It should be pointed out that the function from IOB tag pairs to brackets also takes the OI pair into account, which can be produced by the tagger even though it is strictly speaking not a correct IOB2 sequence.

\[
\begin{array}{c}
OI \mid \\
OB \\
IO \\
BO \\
IB \mid \\
BB \mid \\
II \\
OO \\
BI
\end{array}
\]

Table 6.13: Associating IOB tag pairs with opening and closing brackets.

After having computed the probabilities for all individual IOB tags at two adjacent positions in the sentence, probabilities of sequences of two tags are computed by multiplying separate tag probabilities. The probabilities of pairs of tags are then combined to compute the probabilities of brackets between the two adjacent positions: the probability of a particular type of bracket between positions \(i\) and \(i+1\) is the sum of the probabilities of all pairs of IOB tags \(t_i, t_{i+1}\) that are associated with that type of bracket according to table 6.13, as shown for the different types of brackets in the following four equations.

\[
\begin{align*}
P(\square) &= P(OI) + P(OB) \\
P(\{) &= P(IO) + P(BO) \\
P(\}\) &= P(IB) + P(BB) \\
P(\_) &= P(II) + P(OO) + P(BI)
\end{align*}
\]

The four bracketing options are ranked by their probability and the most likely option is selected. In ranking, the probability of the event that no
Chapter 6. Reducing structural ambiguity

Bracket is assigned receives an extra weight $\tau$. (After applying the extra weight, the probabilities do not need to be normalized, since we are only interested in the ranking and not in the probabilities themselves.) Varying the weight $\tau$ leads to different trade-offs in precision and recall. For instance, a large value for $\tau$ means the probability for “no bracket” becomes higher, leading to lower recall and higher precision. In this application interest is specifically in precision since the parser should not be hindered by too many incorrect brackets. For a given position the bracket $b$ is selected as in equation 6.7 from the set $B$ defined in equation 6.6. (In equation 6.7, probabilities are multiplied with a factor $\omega_x$ that is computed in equation 6.8.)

$$B = \{ [ , ] , - , ] [ \} \quad (6.6)$$

$$b = \arg \max_{x \in B} P(x)\omega_x \quad (6.7)$$

$$\omega_x = \begin{cases} 1, & \text{if } x \neq - \\ \tau, & \text{if } x = - \end{cases} \quad (6.8)$$

In the above approach, a closing bracket immediately followed by an opening bracket (\texttt{[ ]}) is treated as a separate case. Before deciding on this approach, a strategy was tried in which opening and closing brackets were assigned independently by comparing each of them to “no bracket”. However, this would often lead to an incorrect assignment of both brackets at the same time. For instance, if an opening bracket is likely, then “no bracket” is unlikely: as a side-effect, the probability of the closing bracket may end up slightly above the probability of “no bracket” and a closing bracket is incorrectly assigned. This problem is avoided by treating the occurrence of two brackets as a separate case.

### 6.5.3 Stand-alone innermost NP bracketing

**Method and data**

In order to see the effect on precision and recall when varying $\tau$, a stand-alone experiment is run. The tagger is used to assign innermost NP brackets to a sentence after which these are compared to the brackets that were assigned to the same data by Alpino. The data used is the year 1999 of the Parool newspaper to which POS tags and innermost NP brackets have been assigned using Alpino. The same data was used in section 6.5.1; training data is 574,917 sentences (9.0 million words) and test data is 500 sentences (8160
6.5. Reducing structural ambiguity

words). In this experiment the naive two-step method, which achieved low accuracy on the IOB tags, was not considered.

Results

The results in terms of precision for different levels of recall are shown in figure 6.1. The results were attained by using different values for $\tau$: the points on the right side of the graph, corresponding to high recall rates, were produced with low values for $\tau$, making it easier for brackets to be added. The combined model offers the highest precision for a recall of up to about 70%; for higher recall rates the extended two-step method offers higher precision. The naive combined model is clearly performing worse than the other methods.

![Figure 6.1: Levels of precision attained for different levels of recall in recognizing innermost NP brackets, comparing three chunk tagging methods.](image)

On the left hand side of the graph, lower recall rates can be seen to correspond to lower precision rates in the case of the naive combined and the
Chapter 6. Reducing structural ambiguity

extended two-step method, where one would expect to see higher precision rates as only the brackets survive that are assigned very high probabilities. Although it is not completely clear why this unexpected behavior occurs, it is probably related to the following problem. In the case of the extended two-step method and the naive combined method, the different information sources (chunk tags and POS tags) have been combined into one type of n-gram. After tagging a word, it is often the case that no information is available for one or more of the IOB tags, as no tag containing the particular IOB tag was assigned to that word.

An example is given in table 6.14, showing how the naive combined method can lead to a chunk tag being left out if this tag was not seen together with the word and POS tag(s) at hand in the training data. This is to be compared with table 6.15, which shows that in the case of the combined method, all three chunk tags are always present as the three possible context values.

<table>
<thead>
<tr>
<th>( w_{i-1} )</th>
<th>( w_i )</th>
<th>( w_{i+1} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>0_tag1</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>I_tag1</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.14: Example of tag assignment for the naive combined method: chunk tag B was not seen together with POS tag tag1 and word \( w_i \) in the training data, and therefore there is no tag B_tag1, leaving chunk tag B unrepresented.

<table>
<thead>
<tr>
<th>( w_{i-1} )</th>
<th>( w_i )</th>
<th>( w_{i+1} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>0_tag1</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>I_tag1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B_tag1</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.15: Example of tag assignment for the combined method: even though chunk tag B was never seen together with POS tag tag1 and word \( w_i \), B is still considered as one of the three possible context values.

Smoothing of bracket probabilities involving a missing IOB tag is performed by using a default low probability for the missing tag. (This probability was determined by inspecting probabilities assigned in tagging a separate test set, and selecting the lowest probability that was encountered.) At low recall rates the lack of more precise information, as provided by separate n-gram types in the combined model, might be the cause of the low bracket precision.
6.5.4 Supplying brackets to the parser

Following on the stand-alone tests in the previous section it will now be investigated to what extent the assigned innermost NP bracketings, when passed on to the parser and using different levels of recall and precision, can be of help in increasing efficiency and accuracy by reducing structural ambiguity.

Method and data

The chunking models are trained on the Parool 1999 training data (9.0 million words). Using all of the chunking methods except naive two-step, innermost NP brackets are assigned by the tagger to a set of 500 test sentences (8.5 thousand words) taken from the Dutch newspaper Trouw. These bracketed sentences are then used as input to the parser. The experiment is run repeatedly using different values for $\tau$, the extra weight assigned to “no bracket”, to achieve different levels of bracketing precision and recall in the input to the parser.

Results

Figure 6.2 contains the results in terms of parser accuracy and efficiency when parsing the sentences that contain innermost NP brackets. The vertical axis represents the parser accuracy and the horizontal axis represents the average time it takes to parse a sentence. The points in the graph are the result of the use of different values for $\tau$. As well as showing results for the use of three chunking methods, the graph also shows the performance when not using any brackets (“no brackets”), using the correct brackets (“correct brackets”) and using the correct brackets and allowing the parser to assume that every opening bracket has a corresponding closing bracket (“assuming balanced”), which has a positive effect on efficiency and accuracy. (It should be noted that “correct brackets” is used here to mean “brackets used in what the parser considered to be the correct parse”.)

Parsing time, not including time needed by the tagger, was measured on a 3.06 GHz Intel Xeon personal computer. The average time needed by the tagger to assign chunk tags to a single sentence using the three methods was measured separately on a 2.40 GHz Intel Pentium 4 personal computer, and is shown in table 6.16. Average parsing and tagging times were added together, resulting in figure 6.2, in which parser time includes the time needed by the tagger. Note that the points on the graph that represent the results when all correct brackets are available, and the parser is allowed to assume they are balanced, do not take time needed for tagging into account.
Chapter 6. Reducing structural ambiguity

Figure 6.2: Parser performance in terms of accuracy and average parsing time on input sentences to which innermost NP brackets have been assigned using different chunking methods and varying the extra weight $\tau$ for “no bracket”.

The results when using the correct brackets show that in principle an interesting improvement in efficiency without loss of accuracy is possible through the use of NP chunk brackets, but the present methods in their current implementation do not have this effect. (The increase in accuracy when using the correct brackets can be ascribed to the parser’s use of a beam search for the best parse, in combination with the restriction of parsing possibilities at an early stage as applied through the brackets.) Comparing the three methods, it can be seen that the combined method outperforms the two methods that use an extended tagset: unlike the naive combined method, it achieves a level of accuracy that is close to the accuracy of the system without a filter, and unlike the extended two-step method it does so without a large decrease in efficiency.
Table 6.16: Comparing three chunking methods by the average time it takes for the tagger to tag one sentence.

<table>
<thead>
<tr>
<th>method</th>
<th>tagger time (msec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>naive combined</td>
<td>205</td>
</tr>
<tr>
<td>combined</td>
<td>256</td>
</tr>
<tr>
<td>extended two-step</td>
<td>664</td>
</tr>
</tbody>
</table>

6.6 Conclusion

Although baseNP chunking performance using the HMM tagger and the extended two-step method is comparable to results published for other methods applied to the standard WSJ data set for chunking, the use of the chunker to reduce structural ambiguity for the Alpino wide-coverage parser through recognition of innermost NP brackets does not result in an improvement in parsing performance. Based on the results presented on this chapter, a number of problems and possible ways of improving the described technique can be identified.

The stand-alone test reported on in section 6.5.1 showed that the accuracy attained on innermost NP tags, using the Parool data, was not very high in comparison to the result on baseNP tags, using the WSJ data. Despite the fact that these are different data sets, it might indicate that innermost NP chunks are harder to recognize than baseNP chunks. The difference in the definitions of both types of chunks also points in this direction. Consider the following two example sentences in which innermost NP chunks have been marked with brackets. In the sentence in example (4), one of the innermost NPs is *de boerderij* (“the farmhouse”). However, the same phrase *de boerderij* is not an innermost NP in example (5), as in that case, the phrase is part of a larger NP of which *boer Bob* is the innermost NP.

(4) [De eend] landde op het dak van [de boerderij]
    [The duck] landed on the roof of [the farmhouse]

    On the farmhouse of [farmer Bob] landed [a duck]

In deciding whether an NP is an innermost NP, its left and right context has to be taken into account. Deciding whether a phrase is an NP chunk is easier since a simple noun phrase such as *de boerderij* is always an NP chunk. Therefore further research in this direction can be aimed at different types of chunks, possibly involving modifying the parser’s grammar so that brackets indicating (for example) NP chunks are useful in parsing. This is
currently not the case, as brackets used by the grammar (such as innermost NP brackets) are not identical to NP chunk brackets.

It was shown in figure 6.2 that the combined method outperformed the other methods in supporting the parser, considering a trade-off between efficiency and accuracy. However, a problem in deciding which brackets to select based on the assigned IOB tags was encountered for the naive combined and extended two-step methods in section 6.5.3. Perhaps an improvement could be attained by using a different approach to selecting brackets.

Another point concerns the kind of information about bracketing that is passed on to the parser. Currently a strategy is used in which the tagger indicates which positions in the sentence must contain a bracket. This strategy is partly a result of the fact that the training data provided by the parser indicates where brackets are located. However, a different approach could be tried in which the tagger indicates positions that may not contain brackets. The same strategy was used in chapter 4, where only very unlikely POS tags were removed during lexical analysis of a given word, allowing the use of all remaining POS tags; in the case of brackets, this means excluding only those brackets that are very unlikely and allowing brackets on all other positions.

Finally, the extended model used in chapter 5 and used again in the current chapter could be experimented with. As pointed out in section 6.4.2, the model might be less appropriate for chunking than for the task it was originally used for in chapter 5.