Headline generation for Dutch newspaper articles through transformation-based learning

Daniël J.A. de Kok

August 2008

Master's Thesis
Information Science
Faculty of Arts
University of Groningen

Supervisor: dr. G.J.M. van Noord
Reader: dr. G. Bouma

Contents

Chapter 1

Introduction

1.1 Text summarization

The digital age has given us access to an increasing amount of information - historical documents are being digitized, and new information is produced at an astonishing rate. This flood of information can even lead to a state that is sometimes referred to as *information overload*, which Wikipedia defines as:

> Information overload refers to the state of having too much information to make a decision or remain informed about a topic.

Text summarization can provide a partial solution to this problem. By giving short, but accurate summaries of a text a reader can quickly select texts that are interesting and capture the relevant information given by that text.

Not surprisingly, text summarization is currently an active field of research. Traditionally, this task has been performed by selecting and concatenating salient sentences from a text. However, this often leads to summaries that are not very consistent or fluent. In recent years wide-coverage parsing and increased computational power has opened doors for more sophisticated methods for the summarization of texts.

1.2 Headline generation

An important area of research within text summarization is sentence compression. The goal of sentence compression is to shorten a given sentence to a prespecified length. For instance, if we were given the task to shorten the following sentence[1]:

[1] http://news.bbc.co.uk/2/hi/asia-pacific/7498849.stm

> Senior diplomats are meeting in Beijing to thrash out the next move in the long-running mission to end North Korea's nuclear ambitions.

This sentence can be shortened to nearly half its length, without a significant loss of meaning:

> Senior diplomats are meeting in Beijing to end North Korea's nuclear ambitions.

Sentence compression is useful for various tasks such as the extraction of core semantics of a sentence, or for generating short headlines from salient sentences. While headline generation can be useful on its own, it's also a well-defined task for researching and improving sentence compression: there is a wealth of training material available in the form of newspaper or magazine headlines, and it is a task that can relatively easily be verified by humans. For this reason, this thesis focuses on the headline-generation task within sentence compression.

Practical use can include generation of headlines for large text collections that are being digitized. While it is unconceivable that a machine will take over this work from newspaper editors, it can be useful for digitized newspaper articles. While they already have headlines, some articles have very short (e.g. one or two word) headlines. They were never written to be used in a digital context, where we quickly skim RSS feeds to get the latest BBC news, or search engine summaries of web pages.

1.3 Approach

Various different methods have been proposed in the past to generate headlines for newspaper articles. Virtually all of the methods developed in recent years make use of parse trees of newspaper article sentences and newspaper headlines. Usually, the first sentence of an article is selected, and words or constituents that are deemed unimportant are removed. This process is often referred to as *trimming*. Like many other problems in computational linguistics, such as part-of-speech tagging, most methods can be categorized in statistical data-driven and (rule-based) linguistic methods. Well-known approaches in both categories are respectively the noisy-channel model for sentence compression, and the hedge trimmer.

In other areas of research, such as part-of-speech tagging or chunking, transformation-based learning has proven to be a good method that combines advantages of purely statistical methods with rule-based methods. In this thesis, I report on a modified transformation-based learning system for sentence compression, or more specifically, newspaper headline generation. My hypothesis is that transformation-based learning can be used reliably

for headline generation. Additionally, I expect that the rules created by this system mirror the linguistic insights that have fueled systems with hand-written rules such as the hedge trimmer. As a result, a tranformation-based learning system can provide good insights in what elements of a sentence don't contain viable information, and perform the task of headline generation well.

During the project, it quickly became clear that no useful training material was readily available for making a system that could generate Dutch headlines. For this reason, and the lack of time to make a hand-written headline corpus, a procedure for automatic generation of good training material was also devised.

1.4 System overview

While the transformation-based learning system for headline generation will be described in detail in this thesis, this section will provide a quick overview of the system.

The system learns rules for headline generation through a process named *transformation-based learning* (TBL). TBL tries to find transformation rules that are applicable to rewrite sentences to their target form (headlines). During a learning cycle, all candidate rule are scored, and the candidate rule with the highest score is the rule that is learnt during that cycle. This process is repeated until a certain score threshold has been reached. Rules are learnt from parse trees of uncompressed sentences and the words and word roots of correct headlines. The resulting rules can then be applied to the parse tree of a full sentence to generate a headline.

The sentence parse trees were generated with the Dutch Alpino parser and grammar, which produces dependency trees. Alpino dependency trees will be discussed in more detail in section 5.1.

Besides implementing the main learning algorithm, which is comparable to that used in other TBL systems, a few specific questions required answering:

- What kinds of rules should be supported? For instance, should the system only use deletion rules, or also rules that allow for changing the order of tree nodes?
- What kinds of conditions should be available for rules? A condition restricts the application of a rule to a certain context. E.g. conditions could make the application of a rule depend on certain attributes of the affected node.
- How should rules be scored? This question is twofold: firstly, metrics are required to determine if the application of a rule to a sentence parse

tree had a positive or negative effect. Secondly, it should be decided what scores should be attached to positive and negative effects of the application of a rule on a tree.

Most of the existing systems have requirements with respect to training (and testing) material that only make subtree deletions useful. For instance, the hedge trimmer has one rule that can make a subtree node the tree root, but all other rules use some form of deletion[2]. Due to the looser restrictions that were applied to the training material for this thesis, it's useful to allow for other types of rules as well.

The types of conditions that are used, are partly inspired by the conditions used by the hedge trimmer. They mostly center around attributes of nodes in the neighborhood of the node to which the rule is to be applied, such as parents, siblings, and children. Since the Alpino parser provides some very useful attributes, conditions are not necessarily dependent on e.g. just the name of a constituent.

To determine whether the application of a rule had a positive or negative effect, some existing measures were tested. While it may seem natural to use a measure that compares a candidate compressed tree with the parse tree of a correct headline, headlines are often not strictly grammatical, making it hard to make a good parse. For this reason, it seemed more appropriate to use a function that compares the candidate headline and the good headline by comparing words and their order. If a candidate headline has virtually the same words and same order as the correct headline, it's likely to be a good headline. Some methods from the ROUGE test suite [1] are good candidates for providing such a function. Per rule type improvement/penalty points were used to rate the effect a rule had on a tree.

1.5 Thesis overview

Besides this chapter, the rest of this thesis is divided in the following chapters:

- Chapter 2 describes some prior research in the area of headline generation. Two major trimming methods are described (a noisy-channel model trimmer, and the hedge trimmer), and their strengths and weaknesses are identified.

- Chapter 3 gives an overview of transformation-based learning method for learning rules for various transformation tasks. It also describes how transformation-based learning is used in part-of-speech tagging, and how it could be used for headline generation.

[2] Of course, selecting a node, and making it the new tree root is in effect also deletion

- Chapter 4 describes the methodology used for extracting training and testing material from a newspaper corpus.
- Chapter 5 gives a detailed description of the transformation-based learning system that was written for headline generation during this thesis project.
- Chapter 7 discusses some interesting implementation details of the headline generation system.
- Chapter 6 describes the ROUGE metrics for the evaluation of text summarization, and motivates the use of ROUGE-SU in rule learning, and the use of ROUGE-L for evaluating the system.
- Chapter 8 summarizes the results of headline generation using this system, both by employing ROUGE-L and human inspection.
- Chapter 9 analyses the fitness of transformation-based learning for the task of headline generation, and discusses possible future improvements.

Chapter 2

Prior research

In recent years, sentence compression has been an area for active research. This research has been driven by the urge to move text compression beyond merely the selection of salient sentences. Most recent approaches make use of parse trees that are generated by modern, wide coverage parsers. And analogous to many other fields of research, both statistic and linguistic approaches have been suggested. Two well-performing representatives of both approaches are respectively the noisy-channel model and the hedge trimmer. This section will provide a quick overview of both methods, and identify their weaknesses.

Other recent work includes a discriminative large-margin framework by McDonald [2] and a decision-based trimmer that is also proposed by Knight and Marcu [3]. A simpler but faster method that relies on pattern-based removal is described by Conroy et al. [4].

2.1 Noisy-channel model

2.1.1 Introduction

The noisy channel model is a widely-used statistic model for trying to reconstruct data that has possibly been distorted by transmission over a noisy channel. This can also be used to model sentence compression, by regarding the compressed sentence as the original data, and the uncompressed sentence as the original data with additional noise. In other words, the task of sentence compression is reformulated as the discovery of original data in the longer, noisy data.

The noisy-channel model for sentence compression as described by [3] uses parse trees obtained from a Probabilistic Context-Free Grammar (PCFG). As such, probabilities for production rules can be used as probabilities within the noisy-channel model.

When a sentence t is encountered, we want to find a short compressed string s that maximizes $P(s \mid t)$. Through Bayesian inversion, we can re-

define this to the maximization of $P(s) \cdot P(t \mid s)$. The model that defines $P(s)$ is called the source model, and the model that defines the addition of noise, or $p(t \mid s)$, is the channel model. The next sections will describe both models in more detail.

2.1.2 The model

The noisy channel model consists of two models: the *source model* and the *channel model*:

Source model

The source model assigns a probability to every possible compressed sentence. [3] attributes two characteristics to a probable compressed sentence: it should have a normal-looking parse tree, and normal-looking word pairs. The first characteristic is observed through the probability of the parse tree of the sentence, this is the normal PCFG probability that can be obtained through multiplication of applicable rules. The probability of word pairs within the compressed sentence is calculated through word bigram probabilities. The parse tree and bigram probabilities are then multiplied.

To cite a small example from [3], consider the following tree s:

```
S(NP John)
 (VP (VB saw)
     (NP Mary))
```

The probability of the tree s can be calculated in the following manner (the authors use the *EOS* marker at the beginning and end of the sentence):

$$\begin{aligned} P_{tree}(s) \quad = \quad & P_{cfg}(TOP \rightarrow S \mid TOP) \cdot P_{cfg}(S \rightarrow NP\ VP \mid S) \\ & \cdot P_{cfg}(NP \rightarrow John \mid NP) \cdot P_{cfg}(VP \rightarrow VB\ NP \mid VP) \\ & \cdot P_{cfg}(VP \rightarrow saw \mid VB) \cdot P_{cfg}(NP \rightarrow Mary \mid NP) \\ & \cdot P_{bigram}(John \mid EOS) \cdot P_{bigram}(saw \mid John) \\ & \cdot P_{bigram}(Mary \mid saw) \cdot P_{bigram}(EOS \mid Mary) \end{aligned}$$

Channel model

The channel model calculates the probability $P(s \mid t)$ for a pair of strings $< s, t >$, where s is the original sentence, and t the same sentence with the accumulated noise. The addition of noise is performed trough tree expansion. For each internal node, an expansion template is probabilistically chosen based on the node label, and the label of its children. For example, one expansion template for the tree listed above could add a prepositional

phrase with probability $P_{exp}(S \rightarrow NP\ VP\ PP \mid S \rightarrow NP\ VP)$. Another possibility is to keep the node unmodified with probability $P_{exp}(S \rightarrow NP\ VP \mid S \rightarrow NP\ VP)$. When new child nodes are added, a subtree is grown according to the P_{cfg} probabilities of that subtree.

Of course, this is a limited expansion model, since only new subtrees can be added (or deleted viewed from compression). It is not possible to reorder trees, or to modify word inflections.

2.1.3 Training

The noisy-channel model is trained by using a parallel corpus of headlines, and the corresponding first sentence of the newspaper article. The first step that is performed, is finding the syntactic nodes of the two parse trees that correspond. Expansion templates are then extracted from the corresponding nodes. For instance, if the to-be compressed sentence has the following tree:

```
(S (NP ...)
   (VP ...)
   (PP ...))
```

and the corresponding S node from the correct headline has the following tree:

```
(S (NP ...)
   (VP ...))
```

This yields the expansion template ($S \rightarrow NP\ VP\ PP \mid NP\ VP$). The frequency of this expansion can be normalized over all other expansions of $S \rightarrow NPVP$, so that the probability of this particular expansion can be calculated.

The authors have used standard methods to estimate PCFG and word bigram probabilities (from the Penn Treebank and the Wall Street Journal respectively).

2.1.4 Evaluation

The Ziff-Davis corpus was used to train and evaluate the system. This corpus consists of newspaper articles announcing computer products. Many of the articles are paired with human-written abstracts. Sentence pairs were extracted, where the compressed sentence from the abstract is a subsequence of a full sentence from an article. In this manner, they extracted 1067 sentence pairs, of which 1035 pairs were used for training and 32 pairs were used for testing the system.

Additionally, the system was also tested on another corpus, to see how the system performs on a corpus where it was not trained on. This corpus

consisted of the first sentence of 26 articles made available in 1999 on the scientific *Cmplg* archive. The compressed sentences where written by the authors themselves.

In their testing, their noisy-channel based system was compared to a baseline system that makes compressions based on the highest word bigram scores, a decision-based system, and finally the human-written compressions. The decision-based system tries to construct a smaller tree from the parse tree of a full sentence by applying a sequence of shift, reduce, and drop actions as used in extended shift-reduce parsers. The decision-based model is deterministic, it can only produce one compression per input sentence. Since this does not give the same flexibility as the noisy-channel model and the hedge trimmer, this thesis does not give an extensive description of the decision-based model.

Each compression was scored by four judges on a scale from 1 to 5 on how well each system performs with respect to importance (of selecting the most important words from the original sentence), and the grammaticality of the resulting compression. The judges were told that all compressions are machine-generated, and the compressions were given in a random order.

Tables 2.1 and 2.1 show the results from these experiments. The average scores by judges and their standard deviations are shown. The compression rates are also shown (these are the compression rates at which the compression gives the highest score/probability, in the case of the decision-based method there is deterministic and gives only one possible compression). Scores for the baseline system are missing for the Cmplg corpus. Some of the sentences in this corpus were very long, and the baseline model could not produce compressions.

Table 2.1: Results of the baseline and noisy channel models.

Corpus	Avg. orig. length		Baseline	Noisy-channel
ZD	21 words	Compression	63.70%	70.37%
		Grammaticality	1.78 (1.19)	4.34 (1.02)
		Importance	2.17 (0.89)	3.38 (0.67)
Cmplg	26 words	Compression	-	65.68%
		Grammaticality	-	4.22 (0.99)
		Importance	-	3.42 (0.97)

Both the noisy-channel model, and the decision-based model perform well compared to the baseline model. When used on a different type of corpus, the quality of the compression of the noisy channel model declines smoothly, while the decision-based model performs much worse.

Table 2.2: Results of the decision-based model and human summarization.

Corpus	Avg. orig. length		Decision-based	Humans
ZD	21 words	Compression	57.19%	53.33%
		Grammaticality	4.30 (1.33)	4.92 (0.18)
		Importance	3.54 (1.00)	4.24 (0.52)
Cmplg	26 words	Compression	54.25%	65.68%
		Grammaticality	3.72 (1.53)	4.97 (0.08)
		Importance	3.24 (0.68)	4.32 (0.54)

2.1.5 Conclusions

The noisy-channel model has the useful characteristics that most statistic models in language processing have: they can be employed without much manual labor, and can easily be retrained using other corpora. It also shares a common downside: the resulting training data provides relatively little insight, and does not have the clarity of e.g. rule-based systems. Besides that, the noisy-channel model that Knight and Marcu use only allows for deletions, and is not easily expandable to allow for other transformations.

2.1.6 Model extensions

Recent research has proposed extensions to the model described by Knight and Marcu to address some of its shortcomings. For instance, Unno et al. have extended the noisy-channel model with a maximum-entropy model that allows for incorporation of more specific characteristics to decide which node should be removed, such as the mother node, sibling nodes, and the depth of the node [5]. Additionally, they have shown a bottom-up method for better matching of parse trees of uncompressed sentences and reference compressions during the training phase.

2.2 Hedge trimmer

2.2.1 Introduction

The hedge trimmer generates headlines for newspaper stories using linguistically-motivated rules [6]. These rules remove constituents from the parse tree of a sentence until a length threshold has been reached.

The first sentence of an article is used to generate the headline for an article, because the authors found that in manual creation of headlines the majority of headline words were extracted from the first sentence. Depend-

ing on the article corpus, the authors found that on average roughly 51% to 87% of the headline words were picked from the first sentence of an article.

The trimming rules and conditions were derived from manual inspection of human-written headlines for 73 articles from the TIPSTER corpus. Afterwards, some insights from this initial study were confirmed through automatic analysis of 218 human-written headlines.

2.2.2 Heuristics

The heuristics for the hedge trimmer can be divided in three steps:

1. Choose the lowest leftmost S with NP, VP
2. Remove low-content units
3. Iterative shortening

The first two steps are applied once, the third step is applied iteratively until the requested sentence length threshold is reached.

Choose the correct S subtree

During the first step, the lowest leftmost S subtree with NP and VP constituents is chosen. This step relies on the *projection principle* [7]. In human-generated headlines predicates always projected a subject, so the application of the hedge trimmer should also result in sentences that conform to this rule. The following example is given by the authors, bold-faced material is retained, italic material is eliminated:

[S **[S [NP Rebels] [VP agree to talks with government]]** *officials said Tuesday.]*

If the requirement of this step can not be fulfilled, the lowest leftmost S node is selected. If there is no S node, the root of the parse tree is selected.

Remove low-content nodes

During the second step, some low-content nodes are removed. The simplest low-content nodes that are removed are the determiners *the* and *a*. Other determiners, such as negative determiners and quantifiers are not removed, because they can provide vital information that should not get lost during headline generation.

Additionally, the authors found that time expressions can be removed. Although they do provide content, they are often not necessary to represent the who/what content of the story. For instance, “the Democratic primary

season finally draws to a close Tuesday", can be shortened to "the Democratic primary season finally draws to a close", and still cover the content of an article. However, examples where the time expression is important are not unthinkable.

The hedge trimmer removes time expressions by marking them first, and then removing subtrees having the structure *[PP ... [NP [X] ...] ...]* and *[NP [X]]*, where X is a time expression. These removals can lead to ungrammatical sentences, but the authors say that this often pans out, because the ungrammatical fragments stand a good chance of being removed in subsequent steps.

The following example shows the removal of both determiners, and time expressions:

[Det The] **State Departement** *[PP [IN on] [NP [NNP Friday]]]* **lifted** *[Det the]* **ban it had imposed on foreign flyers.**

Iterative shortening

The last step removes rightmost phrasal nodes iteratively until the length of the sentence has reached the threshold. There are four rules in this step:

1. XP-over-XP removal: in structures of the form *[XP [XP ...] ...]* remove the other children of the higher XP, where XP is NP, VP, or S.

2. Removal of preposed adjuncts: all the human generated headlines ignored the preamble of the story. This is phrasal material occurring before the subject of the sentence. In structures of the form *[S ... [XP ...] [NP ...] [VP ...]]]*, where S is the root node, XP should be deleted.

3. Removal of the deepest rightmost PP backward until length is below the threshold.

4. Removal of the deepest rightmost SBAR backward until length is below the threshold.

The PP and SBAR removal rules are derived from the observation that fragments at the beginning of a sentence tend to have more headline words than the end. Still, these rules can remove vital information, and should be applied as a last resort, and with care. Both rules are applied with a back-off option. This means that if PP removal does not result in a sentence that is shorter than the specified threshold, the tree is restored in the previous state, and SBAR removal is attempted. If this does not cause the threshold to be reached, the PP rule is applied as well.

The following four compressions show the rules described above in action:

1. [**S** *[Det A]* **fire killed** *[Det a]* [**NP** [**NP firefighter**] *[SBAR who was fatally injured as he searched the house]*]]

2. [**S** *[PP According to a now-finalized blueprint described by U.S. officials and other sources] [Det the]* **Bush administration plans to take complete, unilateral control of** *[Det a]* **post-Saddam Hussein Iraq**]

3. [**S More oil-covered sea birds were found** *[PP over the weekend]*]

4. [**S Visiting China Interpol chief expressed confidence in Hong Kongs smooth transition** *[SBAR while assuring closer cooperation after Hong Kong returns]*]

2.2.3 Evaluation

The authors of the hedge trimmer tested their system on two corpora. The first corpus consisted of 100 reference headlines that were generated for 100 AP stories from the TIPSTER collection for August 6, 1990. These 100 stories were not used for previous testing of their system or evaluated by the authors. The other corpus consisted of 2496 manual abstracts for the DUC2003 10-word summarization task as reference compressions for 624 documents used in that task.

The system was evaluated with BLUE, a system for automatic evaluation for machine translation[1]. The system was compared with a noisy-channel model for headline generation. The noisy-channel model was tested on both the first sentence of an article, and the first 60 words. Table 2.3 shows the BLUE scores using trigrams, and the 95% confidence interval for the score.

Table 2.3: Hedge trimmer BLUE results.

	AP900806	DUC 2003
HMM60	0.0997 (0.0322) avg len: 8.62	0.1050 (0.0154) avg len: 8.54
HMM1Sent	0.0998 (0.0354) avg len: 8.78	0.1115 (0.0173) avg len: 8.95
HedgeTr	0.1067 (0.0301) avg len: 8.27	0.1341 (0.0181) avg len: 8.50

The hedge trimmer scores slightly better on both data sets, but the authors the results are not statistically significant.

The authors contend that automatic evaluation does not show the difference in quality between the systems. Each of the 100 headlines (per

[1]See chapter 6 for more information about BLUE.

system/variation) that was generated for the 100 AP stories extracted from the TIPSTER corpus was evaluated by a human. Each headline was given a score of 1 to 5. The average score for the HMM hedger was 3.01, with a standard deviation of 1.11. The hedge trimmer had an average score of 3.72 with a standard deviation of 1.26. Using a t-score this difference is significant with a confidence greater than 99.9%.

2.2.4 Conclusions

In addition to good performance, the hedge trimmer uses sound linguistic underpinnings that most statistical headline generation systems miss. It uses rules that are linguistically motivated, and the authors of the hedge trimmer show that there is also statistical evidence to support their choice of rules.

Unfortunately, the hedge trimmer was not applicable as-is. The dependency trees produced by the Alpino parser differ very much from the parse trees generated by the parser that the authors used[2]. Additionally, Alpino does not mark time expressions analogous to the BBN IdentiFinder, making it hard to delete time expressions without further work.

Of course, it would have been possible to rewrite the hedge trimmer rules for the Alpino dependency grammar. But this is not necessary if we can tackle another problem with the hedge trimmer: the need to manually write (and in this case rewrite) rules for trimming. If trimming rules can be generated automatically, this would be cheaper than the manual labor of designing rules, while still producing readable rules that may reflect linguistic intuitions.

[2]The BBN parser.

Chapter 3

Transformation-based learning

3.1 Introduction

Transformation-based learning (TBL) [8] is a widely-used learning algorithm for learning rules for various tasks, such as part-of-speech tagging, unknown word guessing, and noun phrase chunking. This chapter will give a short introduction to TBL, show an example of how TBL is used for part of speech tagging, and finally describe how the task of learning rules for tree trimming differs from traditional TBL tasks.

In summary, TBL tries to find rules to successfully rewrite a dummy (imperfect) corpus to a 'gold standard' corpus that is known to be correct. The idea is that if the dummy corpus resembles raw material outside the training corpus, and the gold standard corpus resembles the ideal transformation of the raw material, the same rules can be used outside training conditions.

An important question is how rules are learnt. Rules change units within a corpus, based on their contexts. Learning different contexts without predetermined boundaries would create an impractical number of candidate rules. For this reason, most TBL systems use rule templates to generate rules. During the first step of the learning process, all possible rules given the rule templates are gathered, resulting in a large set of candidate rules.

The second step gives a score to all possible rules. This can be done by looking what effect the rule has if it were applied to the corpus. At each spot in the dummy corpus the rule is applied, it can correct an error, introduce an error, or transform an error into another error. The score of a rule is determined by subtracting the number of errors that were introduced from the number of errors that were corrected. The rules can then be ordered by decreasing score, and the rule that had the highest score is stored in a list.

After determining the best-scoring rule, this rule is applied to the dummy corpus. Since it (given a proper threshold) made more corrections than it did

introduce errors, the transformation brought the dummy corpus closer to the gold standard corpus. After applying the rule, all steps are repeated until a predefined threshold is reached. The result is a set of rules that rewrite were proven to rewrite the dummy corpus to the gold standard corpus best.

It should be noted that in most traditional TBL tasks units are only changed. Since this is non-destructive, it's often not necessary to give punishment for errors more than corrections.

3.2 Transformation-based learning in part-of-speech tagging

3.2.1 POS tagging

One application were TBL gained a lot of popularity is part-of-speech (POS) tagging, most prominently through the Brill tagger [8]. POS tagging is the task of assigning lexical categories (tags) to the words in a sentence. For instance, the sentence "The cat is on the mat." could be tagged in the following manner, with the widely-used tag set from the Brown corpus[1]:

```
The/at cat/nn is/bez on/in the/at mat/nn ./.
```

Here, words and tags are separated with a forward slash. *at* is an article, *nn* is a noun, *in* marks a preposition, and *bez* is the verb 'to be' in present tense, third person singular.

POS tagging can be performed fairly accurately by counting word and tag frequencies from a corpus, and assigning the tag that maximizes the $P(w|t)$, which can be calculated with:

$$P(w|t) = \frac{f(w,t)}{f(t)} \tag{3.1}$$

Using this simple method for tagging, an accuracy of over 93% was observed for known words[2]. Unfortunately, this is not good enough due to two problems. First of all, even with sophisticated techniques, the accuracy for tagging of unknown words is nearly always worse. So, further work is required to improve the tagging of unknown words. Secondly, such accuracies are often not good enough for applications that rely on the results of POS tagging. For instance, parsers generally perform much better when the POS tagging was performed more accurately.

[1] http://www.scs.leeds.ac.uk/ccalas/tagsets/brown.html

[2] Known words are words that were seen in the training material.

3.2.2 Improvement through transformation-based learning

To see how TBL can help to improve tagging, let's have a look at an example of a typical tagging error:

```
Jack/np had/hvd a/at drink/vb ./.
```

Here, the word *drink* is tagged as a verb, where it should have been tagged as a noun. This error arises, because $P(drink|vb)$ is larger than $P(drink|nn)$. As humans, we can easily spot that by looking at the context - *drink* it is preceded by an article, this in conjunction with the verb *had* gives us enough reason to believe that *drink* is a noun. Or, we could formulate a simple rule (following the syntax of the aforementioned Brill tagger)

```
vb nn PREVTAG at
```

which states that a *vb* tag should be replaced with *nn* if the previous tag is *at*. And this is exactly what the learning phase of the Brill tagger does, it generates such correction rules through TBL.

As mentioned previously, TBL normally uses rule templates to generate rules. Table 3.1 shows some of the templates that are used by the Brill learner:

Table 3.1: Examples of Brill tagger rule templates

A B PREVTAG C	Change tag A to B if the previous tag is C
A B PREV1OR2TAG C	Change tag A to B if one of the previous tags is C
A B NEXTTAG C	Change tag A to B if the next tag is C
A B NEXT1OR2TAG C	Change tag A to B if one of the next two tags is C
A B SURROUNDTAG C D	Change tag A to B if tag A is (directly) surrounded by C and D

The general TBL procedure described earlier in this chapter is also applied in the Brill learner, where the dummy corpus is the training corpus tagged with the tags that maximize $P(w|t)$. Applying such transformations to the Brown corpus with the Penn treebank tagset improved known word accuracy from 93.3% to 94.5%, and unknown word accuracy from 74.7% to 80.8% [8].

3.3 Application to trimming

TBL seems to be a fit candidate for learning rules for trimming to generate newspaper headlines. If headlines are generated by selecting important

words from the first sentence of a newspaper article, we can formulate this task in reverse: headlines can be generated by applying transformations to the first sentence of a newspaper article.

Since the words (or roots, as we will see later) in the headline are a subset of those in the first sentence, one of the main operations will in effect be the deletion of words. This is an operation that does usually not occur in traditional TBL tasks, which center around changing units (such as tags). In these tasks it often occurs that a rule changes a unit, but changes it back to the original value given a more specific context. Most transformations involved in headline generation do not have this property and are destructive. This should be taken into account during the scoring of rules.

An additional difference compared to traditional TBL tasks, is that the material on which transformations are performed, namely parse trees, are not one-dimensional. A given node can have a parent, one or more children, and one or more siblings. This requires thought about the order wherein trees can or should be traversed, and what effect operations can have on this traversal. On large training sets this can also lead to a large number of rule candidates being generated.

The topic of using TBL for headline generation will be described in detail in chapters 5 and 7.

Chapter 4

Generation of a headline corpus

4.1 Introduction

Good corpora containing articles and headlines are required for testing headline generation systems. Systems that are not purely based on hand-written rules also require sufficient training material. For instance, Knight and Marcu use over 1,000 full sentence-headline pairs for training their noisy-channel model [3].

Unfortunately, no such corpora are readily available for Dutch. Usually, headline corpora are created through manual labor and verification. Unfortunately, there was not enough time to build a large well-validated corpus for this thesis project. For this reason, it was attempted to generate training material automatically.

Before looking at the actual training material generation procedure, some criteria have to be established to which training material should conform. Previous research seems to use training material which has the following implicit or explicit requirements:

1. The words used in the headline should also occur in the untrimmed sentence, and in the same order as in the untrimmed sentence. Or in other words, the headline should be a subsequence of the original sentence.

2. The headline should capture the essence of the untrimmed sentence.

3. The headline should, in most occasions, be shorter than the untrimmed sentence. Sometimes the sentence is already brief enough to serve as a headline.

The first criterium can be problematic. While it seems fair to restrict headlines to being a subsequence of untrimmed sentence, it's often useful to

change the word order and the inflection of words. For instance, consider the following untrimmed sentence, and an appropriate headline:

```
Untrimmed: Spanje, dat gisterenavond Duitsland in de finale met
           1-0 versloeg, heeft de EK beker gewonnen.
           (Spain, which has beaten Germany in the final with
           1-0, has won the EK cup.)
Trimmed:   Spanje won de EK beker.
           (Spain won the EK cup.)
```

Of particular note here is the change of the past participle *heeft gewonnen* (had won) to the past tense *won*. The word order changes (*won* takes the place of *heeft*), as well as a verb through the change in inflection[1]. Since such changes are common in Dutch, the task can become too artificial if absolutely no changing of word order and inflection is allowed.

For this reason, it was decided to change the first criterium to

1. The roots of the words in the headline should be a subset of the roots of the words in the untrimmed sentence, where 'root' is defined as the word without its inflectional endings.

Of course, this looser criterium makes headline generation harder, since it can not solely rely on the deletion of words.

4.2 Procedure

An easily reproducable procedure was used to generate the training data, that relies on software that can determine the roots (or lemmas) of words, and the availability of newspaper text with headlines.

First, pairs of headlines and first sentences of newspaper articles were extracted. For the experiments described in this thesis, all AD, NRC, Parool, Trouw, and Volkskrant articles from the 2002 Twente News Corpus[2] were used. For convenience, this subset will be named News2002 in this thesis. The headlines and sentences were then parsed using the Alpino parser. Full parses are not required for this step, but Alpino includes the roots of words in the parse trees that are generated. And since the sentence parse trees are used for training and testing later on, it was decided to make full parses at this stage.

These two steps provide many headline-sentence pairs, but a lot of headlines do contain words that are not used in the first sentence, or do not

[1] The system described in this thesis will try to deal with changes in word order. But, as we will see, it turns out to be hard to solve this particular issue without sacrificing the generic nature of a system.

[2] A Dutch corpus containing news paper articles, magazine articles, teletext, and news bulletin auto-cues.

use any of the words in the first sentence. Therefore, pairs that conform to the revised first criterium outlined in the previous section are extracted by treating the headline and first sentence as sets of roots, where the headline should be a subset of the full sentence.

The previous step leaves us with material that is almost ready for training and testing. Some testing showed that including one-word headlines has a detrimental effect on the learning process. Since it is more likely that the roots of a headline are a subset of the roots of a full sentence when a sentence is shorter, the generated corpus consists of many short headlines, including a large number of one-word headlines. 4.1 shows the distribution of headline lengths of the headlines that were generated with this procedure.

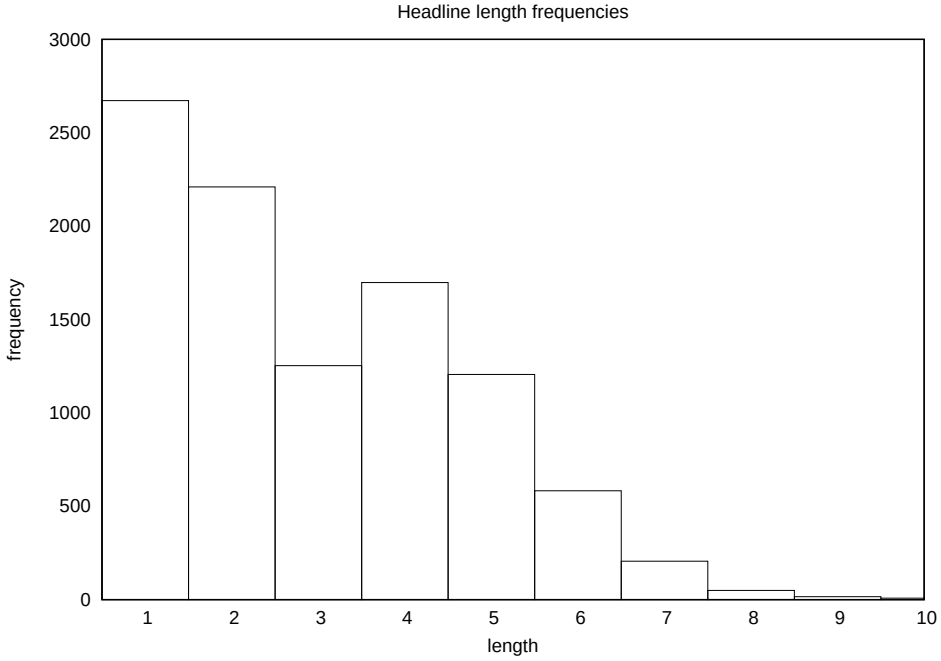

Figure 4.1: Headline length distribution of the 9905 headlines of the News2002 corpus, where headline roots are a subset of the first sentence roots.

The effect of these one-word headlines is that rules that wildly delete subtrees and introduce errors, are often in points compensated by the volume of deletions that are useful for generating one-word headlines. For this reason, one-word headlines were not considered for inclusion in the training data. Deleting one-word headlines left enough material for training and testing the system.

By applying this procedure to the News2002 corpus, 7233 headline/sentence pairs were extracted.

Chapter 5

TBL for trimming

5.1 Introduction

This chapter describes the implementation of the transformation-based learning system that was implemented during this thesis project. The learning component revolves around a loop that finds applicable rules, scores these rules, and applies the rules to the corpus of candidate headlines. This process is repeated until the highest rule score drops below a prespecified threshold.

A rule consists of an action and a set of conditions. An action modifies a parse tree. For instance, an action could delete a node. An action is only applied to a parse tree node, when all the conditions that are associated with a rule are found to be true. Rules are generated using a rule template. A rule template specifies an action, and a set of condition types. An actual rule is generated when a rule template is applied to a parse tree node. This model provides the flexibility to easily create new rule templates, and allows for future additions of actions and condition types.

The input to the learning component are parse trees of the first sentence of a news article and the (correct) headline for the article. No parse is needed of the correct headline, only the actual headline and the word roots. The parse trees for the first sentence are generated with the Alpino parser. Alpino can output parse trees in various formats, such as XML. Figure 5.1 shows the tree that is generated by Alpino for the sentence "Jan ziet de man." ("Jan sees the man."). The XML file representing this structure is shown in listing 5.1.

Various useful attributes are included in the XML representation. Important attributes for this application are[1]:

- **begin/end**: The begin/end positions of the node.
- **cat**: The category of the node.

[1] An extensive description of the attributes, and tools to analyze them can be found at: http://www.let.rug.nl/~vannoord/alp/Alpino/TreebankTools.html

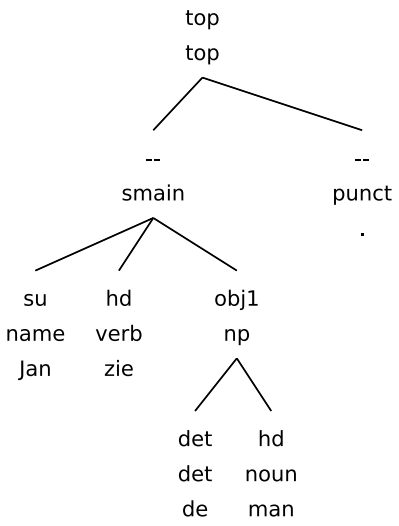

Figure 5.1: A parse tree for the sentence "Jan ziet de man".

- **rel**: The dependency relation of the node.
- **word**: The word/surface form of a leaf node.
- **root**: The root form of a word.

The actual sentences can easily be retrieved from this tree by retrieving all nodes with the *word* attribute, and then ordering them by their positions as indicated by the *begin* attribute.

5.2 Condition generators

As mentioned before, the system uses conditions to determine if a rule is applicable to a node. In a rule template, a condition is only partially specified using a condition generator. A condition generator specifies the condition type and the attribute which a condition checks, but not the actual value of the attribute. All condition generators operate relative to a node, and if it is used on a node, it returns an actual condition instantiated from that node. Currently, there are three types of conditions:

- **HasAttribute**: this condition is true when the node has the specified attribute/value pair. For instance, a condition could say that the *rel* attribute of a node has the value *su*.
- **HasParent**: this condition is true when the parent (or other other ancestor, as defined by the number of steps that should be traversed

Listing 1 XML representation of the parse tree for "Jan ziet de man."

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<alpino_ds version="1.2">
  <node begin="0" cat="top" end="5" id="0" rel="top">
    <node begin="0" cat="smain" end="4" id="1" rel="--">
      <node begin="0" end="1" id="2" pos="name" rel="su"
        root="Jan" word="Jan"/>
      <node begin="1" end="2" id="3" pos="verb" rel="hd"
        root="zie" word="ziet"/>
      <node begin="2" cat="np" end="4" id="4" rel="obj1">
        <node begin="2" end="3" id="5" pos="det" rel="det"
          root="de" word="de"/>
        <node begin="3" end="4" id="6" pos="noun" rel="hd"
          root="man" word="man"/>
      </node>
    </node>
    <node begin="4" end="5" id="7" pos="punct" rel="--"
      root="." word="."/>
  </node>
  <sentence>Jan ziet de man .</sentence>
</alpino_ds>
```

upwards, i.e. 1 for the parent, 2 for the grandparent) has the specified value/attribute pair.

- **HasSibling**: this condition is true when the sibling at a specified position (e.g. *-1* for the sibling left of the current node, or *1* for the sibling right of the current node) has a attribute/value pair.

Each condition type has its own condition generator. Suppose that we created a generator for the *HasAttribute* condition with *rel* as its target attribute, and a condition generator for the *HasParent* condition with *cat* as its target attribute, and apply it to the node with the *np* category of the parse tree described in listing 5.1, this would generate two conditions:

- A *HasAttribute* condition matching nodes with attribute *rel*, having the value *obj1*.

- A *HasParent* condition matching nodes that have a parent with attribute *cat*, having the value *smain*.

5.3 Actions

Actions modify a parse tree node, or a node relative to a parse tree node, when all the rule conditions are satisfied. The system currently knows four different actions, that can be applied to a given node:

- *Delete*: actions of this type delete the node (and all subtrees) when its conditions are satisfied.
- *SwapSibling*: sibling swapping actions swap a node with one of its siblings[2] when its conditions are satisfied. The sibling is specified as a position relative to the node (e.g. *-1* for the sibling left of the current node).
- *MakeRoot*: actions of this type make the node, if it satisfies the associated conditions. the root of the parse tree.
- *ReplaceParent*: actions of this type replace the parent of the node with the node itself when its conditions are satisfied.

In the following sections, these actions will be described with some examples.

5.3.1 Deletion actions

Deletion actions are by far the most frequent actions produced during the learning process, since they embody the main task of the headline generation process, deleting words. For instance, the highest scoring rule that was generated from the training corpus removes punctuation (periods):

```
DeleteAction ANY_NODE 0
  [HasAttributeCondition "word" = "."]
  [HasAttributeCondition "pos" = "punct"]
```

The first element of this rule specifies the name of the action, the second the scope of the action and rule, which will be described later. The third element (in this case *0*) determines what node should be deleted. *0* will delete the node itself, *1* its parent, *2* its grandparent, etc. All bracketed elements are the conditions that are associated with this action. So, here a node is deleted if it holds a period character with the *punct* part-of-speech tag.

[2] Nodes that share the same parent.

5.3.2 Parent replacement

The *ReplaceParent* action occurs fairly often in more specific rules. We will look at an example where this action is useful. In this example a multi-word unit within a subject can represent the whole subject in the compressed sentence. The following rule makes this transformation:

```
ReplaceParentAction RIGHTMOST_NODE
  [HasParentCondition 1 "rel" = "su"]
  [HasParentCondition 1 "cat" = "np"]
  [HasAttributeCondition "rel" = "app"]
  [HasAttributeCondition "cat" = "mwu"]
```

For example, consider the sentence "De Zweed Magnus Wislander, de speler van de eeuw, verlaat Bundesligaclub THW Kiel." ("The Swede Magnus Wislander, the player of the century, leaves Bundesliga club THW Kiel."). The parse tree for this sentence is shown in figure 5.2. In compressed variants of such sentences, the determiner and head noun of the subject can often be deleted, keeping a multi-word unit, that is often a name. In this case, there is also an additional NP that can also be removed. Rather than having a collection of deletion rules that are often less specific, this compression can be performed by replacing the subject node with the multi-word unit. This transformation automatically removes 'de Zweed', and 'de speler van de eeuw', compressing the sentence to "Magnus Wislander verlaat Bundesligaclub THW Kiel."[3] ("Magnus Wislander leaves Bundesliga club THW Kiel.").

5.3.3 Make root actions

The *MakeRoot* action takes a node that satisfies the conditions that are associated with a rule, and makes it the root note of the parse tree (or more specifically, the child of the *TOP* node in Alpino parse trees). This action can be seen as a generalization of the *ReplaceParent* action. *ReplaceParent* could be extended to allow replacement of grandparents, etc., but the *MakeRoot* action allows for making a node at any depth the root node. This action was specifically added to be able to make a template that mirrors the first rule of the hedge trimmer, as we will see in section 5.4.1.

One particular case where this action is very useful, is the following rule:

```
MakeRootAction LEFTMOST_NODE
  [HasAttributeCondition "rel" = "sat"]
  [HasAttributeCondition "cat" = "smain"]
```

[3] In this sample compression the commas are also removed, while this won't actually happen during this transformation, other rules take care of this.

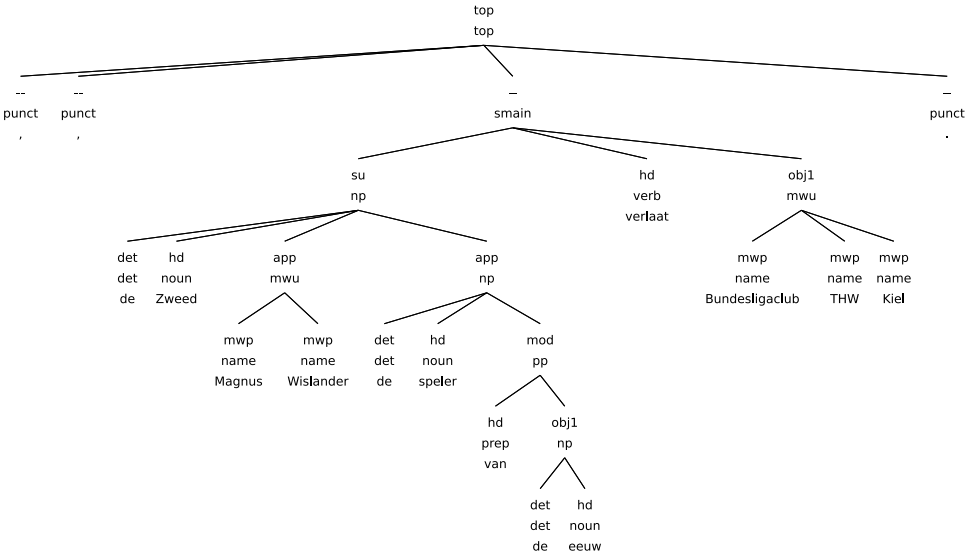

Figure 5.2: A parse tree for the sentence "De Zweed Magnus Wislander, de speler van de eeuw, verlaat Bundesligaclub THW Kiel." (note that a parse failure occurred here: 'Bundesligaclub' should be the head noun of the indirect object'.

Sometimes headlines have some material before the main sentence. Often this is a single word that contains more information about the subject of the article. For instance:

```
Basketbal: Chicago Bulls heeft Jalen Rose van de Indiana Pacers
  overgenomen.
(Basketball: Indiana Pacers have traded Jalen Rose to the
  Chicago Bulls.
```

Alpino parses such cases by subdividing this *discourse unit* relation into a nucleus ("Basketball") and a satellite "Chicago [...] overgenomen.". Sometimes there are even larger nuclei, such as a full prepositional phrase, for instance:

```
Van de Wiener Sangerknabe tot componist van de wereldhit
  'Frankenstein!': HK Gruber is een prettig gestoord fenomeen.
(From the Wiener Sangerknabe to composer of the world hit
  'Frankenstein!': HK Gruber is a pretty crazy phenonemon.
```

In nearly all cases the interesting headline words are in the satellite if it has a *smain* dependency relation, as expressed in the rule shown above. So, this rule will effectively retain the satellite of the discourse unit ("Chicago

Bulls heeft Jalen Rose van de Indiana Pacers overgenomen." and "HK Gruber is een prettig gestoord fenomeen." in the examples above) and remove all other material.

5.3.4 Sibling swapping

The *SwapSibling* action primarily occurs in fine-grained rules. It's sometimes useful to swap constituents to get a more headline-like sentence. For example, consider the sentence "Voor Dennis van Scheppingen is het toernooi van Chennai ook voorbij." (literal: "For Dennis van Scheppingen is the tournament of Chennai also over."). The parse tree for this sentence is shown in figure 5.3. At some point the sentence head verb *is* is removed. Then it can become viable to swap the subject and the *PP* modifier. With some additional removals this can lead to a headline like "Toernooi Dennis van Scheppingen voorbij" (literal: "Tournament Dennis van Schappingen over").

As expected, this is one of the sibling swapping rules that were generated:

```
SwapSiblingAction ANY_NODE
  [HasAttributeCondition "rel" = "mod"]
  [HasAttributeCondition "cat" = "pp"]
  [HasSiblingCondition 1 "rel" = "su"]
  [HasSiblingCondition 1 "cat" = "np"]
```

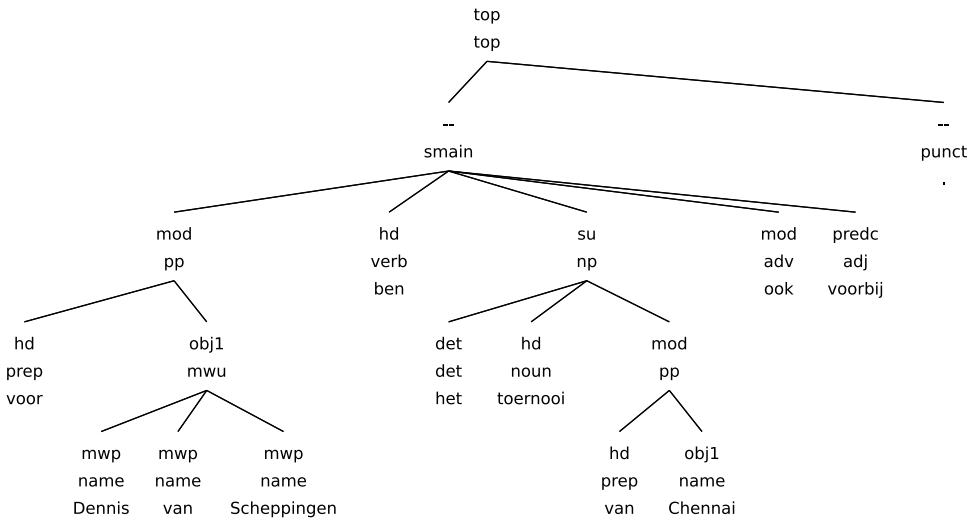

Figure 5.3: A parse tree for the sentence "Voor Dennis van Scheppingen is het toernooi van Chennai ook voorbij".

Swapping is potentially possible in two directions, but swapping with a node on the left can also be expressed with swapping with a node on the

right, seen from the left node. So, for simplicity a *SwapSibling* action always swaps the current node with the sibling on the right.

A limitation of the current *SwapSibling* action is that it is limited to swapping with the sibling that's immediately right of the current node, within the same subtree.

5.3.5 Rule scope

Each rule requires a parameter that specifies the scope of the rule. There are three possible rule scopes:

- *Any*: rules with this scope are applied to any node that satisfies the conditions of a rule. The parse tree is processed breadth-first.
- *Leftmost*: rules with the leftmost scope process the tree in a left-right order. If the rule could be applied to a node, the rule application process is stopped. In effect, the rule only operates on the leftmost node satisfying the conditions.
- *Rightmost*: this scope has the same behavior as the *leftmost* scope, but then for the rightmost node satisfying the rule conditions.

The scope is specified in a rule template, and is used in all rules that were generated with that template.

5.4 Templates

As described in section 5.1, rules are generated from a rule template that specifies an action and a set of condition types. Templates are currently codified in the Java source code of the rule learner, but it would be easy to create a format and a parser for specifying the rules in an external text file.

This is an example of what a rule template can look like:

```
Set<ConditionGenerator> condGens = new
  HashSet<ConditionGenerator>();
condGens.add(new HasAttributeCondition.Generator("pos"));
condGens.add(new HasAttributeCondition.Generator("word"));
RuleTemplate ruleTemplate = new RuleTemplate(RuleScope.ANY_NODE,
    new DeleteAction(), condGens);
```

This is a typical rule template for generating word-based rules. A rule template is constructed from a set of conditions and an action. In this case, there are two condition generators, namely a condition generator for the *pos* attribute of nodes, and a condition generator for the *word* attribute of nodes. Since every parse tree node that represents a word has these

attributes, this template will generate a rule candidate for every word node. The rule discussed in section 5.3 that removed punctuation, was one of the rules generated through this template.

This notation will be used in the examples of rule templates that follow, because the syntax is likely to be familiar to most readers, and they form a direct representation of the actual rule template.

The system was tested with two sets of rule templates: a set of templates that mirrors the rules used by the hedge trimmer, and a set that uses these rules plus some additional rules. The following two sections will describe the hedge trimmer-inspired rule templates, and the extra templates.

5.4.1 Hedge trimmer-like templates

The first set of rule templates is based on the rules used by the hedge trimmer, as described in section 2.2. The rule templates are described in such a manner that the rules used by the hedge trimmer could be generated by these templates. The rule templates that are used are described per hedge trimmer step.

Choose the correct S subtree

The hedge-trimmer selects the lowest leftmost S subtree with NP and VP constituents. This node is made the root of the tree. This system can generate comparable rules, through the *MakeRoot* action. Since Alpino describes nodes both by its category and a dependency relation, and a declarative sentence is indicated by the *smain* category, we can generate the same kind of rules by making a template that has condition generators for the category and relation of the node. So, we can use the following short template to generate rules like this hedge trimmer rule:

```
Set<ConditionGenerator> condGens =
  new HashSet<ConditionGenerator>();
condGens.add(new HasAttributeCondition.Generator("rel"));
condGens.add(new HasAttributeCondition.Generator("cat"));
RuleTemplate ruleTemplate = new RuleTemplate(RuleScope.LEFTMOST_NODE,
    new MakeRootAction(), condGens);
```

Remove low-content nodes

The second hedge-trimmer step is to remove low-content nodes. The first type of low-content nodes are the determiners *the* and *a*, and time expressions. Rules for determiners can be generated with a simple template that looks at the *word* and *pos* attributes of a node:

```
Set<ConditionGenerator> condGens =
```

```
  new HashSet<ConditionGenerator>();
condGens.add(new HasAttributeCondition.Generator("pos"));
condGens.add(new HasAttributeCondition.Generator("word"));
RuleTemplate ruleTemplate = new RuleTemplate(RuleScope.ANY_NODE,
    new DeleteAction(), condGens);
```

To allow for a bit more abstraction, a comparable rule with condition generators for *pos* and *rel* attributes was also added. Besides removing determiners, rules generated with these templates were also very useful to remove punctuation.

Since a tool like BBN IdentiFinder was not readily available for Dutch, we can not mark time expressions. So, instead, the system has to rely on finding words that are used in time expressions that can be removed. The hedge trimmer removes subtrees having the structure *[PP ... [NP [X] ...] ...]* and *[NP [X]]*, where X is a time expression. As in previous examples, we can rely on the condition generators the capture the phrasal structure. To identify and remove phrases, condition generators for the *word* and *pos* attributes were used to identify interesting words, and condition generators for the *cat* and *rel* attributes on parent nodes were used. If all conditions are satisfied, the parent node is removed. The following snippet shows the rule template that is used:

```
Set<ConditionGenerator> condGens =
  new HashSet<ConditionGenerator>();
condGens.add(new HasAttributeCondition.Generator("pos"));
condGens.add(new HasAttributeCondition.Generator("word"));
condGens.add(new HasParentCondition.Generator("rel"));
condGens.add(new HasParentCondition.Generator("cat"));
RuleTemplate ruleTemplate = new RuleTemplate(RuleScope.ANY_NODE,
    new DeleteAction(1), condGens);
```

The additional parameter *1* to the *Delete* action indicates that the node above the current node should be deleted (one level up). A comparable rule template is added that also has condition generators for the *cat* and *rel* attributes of the grandparent. When all the conditions are satisfied, the grandparent of the current node is deleted. These two templates cover the rules that can be generated for removal of phrases containing time expressions.

Iterative shortening

The final step of the hedge trimmer is iterative shortening. Since the system described in this thesis generates a list of rules without iteration, this step can only be approximated. If necessary, the same rule will be generated multiple times. The first possible shortening during this step is XP-over-XP

removal, where in structures of the form *[XP [XP ...] ...]* other children of the higher XP are removed. We can redefine this rule by saying that the lower XP replaces the higher XP, which is covered by the *ReplaceParent* action that was described earlier. A rule template is used with condition generators for the *cat* and *rel* attributes of a node, and its parent:

```
Set<ConditionGenerator> condGens =
  new HashSet<ConditionGenerator>();
condGens.add(new HasAttributeCondition.Generator("rel"));
condGens.add(new HasAttributeCondition.Generator("cat"));
condGens.add(new HasParentCondition.Generator("rel"));
condGens.add(new HasParentCondition.Generator("cat"));
RuleTemplate ruleTemplate = new RuleTemplate(RuleScope.ANY_NODE,
    new ReplaceParentAction(), condGens);
```

It should be noted that XP-over-XP does not occur very often in Alpino dependency trees. For instance, in the sentence "Het vuur overweldigde de brandweerman die gewond was." ("The fire overwhelmed the firefighter who wounded was."), 'brandweerman' becomes the head noun of the NP "de brandweerman die gewond was". Still, in such cases the same material gets deleted through rules that successively deletes the NP modifier (e.g. in this sentence the pronoun is removed first by letting the body of the subordinate clause replace its parent, and afterwards the clause itself is removed). Since the more general rule template described above still fulfills a role, it was decided to include it as well.

The second iterative shortening step of the hedge trimmer removes preposed adjuncts. In structures of the form *[S ... [XP ...] [NP ...] [VP ...]]]*, where S is the root node, XP is deleted. While Alpino generates a different structure, where the preposed adjunct is a modifier of the *VC* and a sibling to a reference to the subject, a comparable rule template can be used. This template uses a deletion action with a condition generator for the *rel* attribute of the sibling of the node, and condition generators for the *cat* and *rel* attributes of the node itself, and its parent:

```
condGens.add(new HasAttributeCondition.Generator("rel"));
condGens.add(new HasAttributeCondition.Generator("cat"));
condGens.add(new HasParentCondition.Generator("rel"));
condGens.add(new HasParentCondition.Generator("cat"));
condGens.add(new HasSiblingCondition.Generator("rel", 1));
RuleTemplate ruleTemplate = new RuleTemplate(RuleScope.ANY_NODE,
    new DeleteAction(), condGens);
```

The last two iterative shortening step is the removal of rightmost *PP* and *SBAR* nodes. This rule is reflected by a rule template based on the deletion

action with a scope on the rightmost node, and condition generators for the *rel* and *cat* attributes of a node:

```
Set<ConditionGenerator> condGens =
  new HashSet<ConditionGenerator>();
condGens.add(new HasAttributeCondition.Generator("rel"));
condGens.add(new HasAttributeCondition.Generator("cat"));
RuleTemplate ruleTemplate = new RuleTemplate(RuleScope.RIGHTMOST_NODE,
    new DeleteAction(), condGens);
```

5.4.2 Extra rule templates

Besides modeling rule templates after the rules used by the hedge trimmer, some additional rule templates that have proven to be useful were added. These rule templates can be subdivided in three categories: deletion conditioned on parent attributes, deletion conditioned on sibling attributes, and sibling swapping. The following sections will describe these rule templates categories.

Deletion conditioned on parent attributes

A rule template that deletes nodes based on the *cat* and *rel* attributes of the current node and its parent was added:

```
Set<ConditionGenerator> condGens =
  new HashSet<ConditionGenerator>();
condGens.add(new HasAttributeCondition.Generator("rel"));
condGens.add(new HasAttributeCondition.Generator("cat"));
condGens.add(new HasParentCondition.Generator("rel"));
condGens.add(new HasParentCondition.Generator("cat"));
RuleTemplate ruleTemplate = new RuleTemplate(RuleScope.ANY_NODE,
    new DeleteAction(), condGens);
```

This template produces rules that are useful for more aggressive trimming, such as the removal of *pp* modifiers of *np*s. For instance, consider the sentence "De Drieklovendam in de rivier de Yangtze in China vertoont scheuren." (Literal: "The Three Gorges Dam in the river the Yangtze in China shows cracks."). There are two PPs in this sentence: "in de rivier de Yangtze in China" and "in China". Since the conditions are generated on the dependency relation besides the category, more fine grained rules are produced. One of the best scoring rules generated from this template is:

```
DeleteAction ANY_NODE 0
  [HasAttributeCondition "rel" = "mod"]
  [HasAttributeCondition "cat" = "pp"]
```

```
  [HasParentCondition 1 "rel" = "app"]
  [HasParentCondition 1 "cat" = "np"]
```

Application of this rule to this sentence removes "in China", compressing the sentence to: "De Drieklovendam in de rivier de Yangtze vertoont scheuren." A subsequent rule also removes the direct *PP* of the subject ("in de rivier de Yangtze"). Such precise rules allow for fine-grained removal, based on the compression rate that is required.

Deletion conditioned on sibling attributes

Two rule templates were added that delete a node based on attributes of its sibling. These templates generate conditions for the *rel* and *cat* attributes of respectively the left and right sibling, and the same attributes for the current node. This template generates rules that become useful when very aggressive trimming is required. For instance, in headlines that consist of just two or three words, the direct object of the sentence head verb is often preferred over the subject of the sentence. So, a rule is generated that deletes the subject if it has a direct object as its sibling:

```
DeleteAction ANY_NODE 0
  [HasAttributeCondition "cat" = "np"]
  [HasAttributeCondition "rel" = "su"]
  [HasSiblingCondition 1 "cat" = "np"]
  [HasSiblingCondition 1 "rel" = "obj1"]
```

The following rule template is used for deletion based on the properties of the sibling on the right of the current node:

```
Set<ConditionGenerator> condGens =
  new HashSet<ConditionGenerator>();
condGens.add(new HasSiblingCondition.Generator("cat", 1));
condGens.add(new HasSiblingCondition.Generator("rel", 1));
condGens.add(new HasAttributeCondition.Generator("rel"));
condGens.add(new HasAttributeCondition.Generator("cat"));
RuleTemplate ruleTemplate = new RuleTemplate(RuleScope.ANY_NODE,
    new DeleteAction(), condGens);
```

A template based on the left sibling was also added.

Sibling swapping

The merits of sibling swapping are discussed in section 5.3.4. A rule template was necessary to operationalize this rule. Since swapping affects two nodes (the node being processed, and its sibling), it is best to add condition

generators on these two nodes. The sibling swapping template uses condition generators for the *cat* and *rel* attributes of the current node and its sibling:

```
Set<ConditionGenerator> condGens =
  new HashSet<ConditionGenerator>();
condGens.add(new HasAttributeCondition.Generator("cat"));
condGens.add(new HasAttributeCondition.Generator("rel"));
condGens.add(new HasSiblingCondition.Generator("cat", 1));
condGens.add(new HasSiblingCondition.Generator("rel", 1));
RuleTemplate ruleTemplate = new RuleTemplate(RuleScope.ANY_NODE,
    new SwapSiblingAction(), condGens);
```

Since the *SwapSibling* action always swaps a node with its immediate sibling on the right side, no rule with mirrored condition generators is required.

5.5 Learning

Transformation rules are learnt in cycles, during each cycle one rule is learnt. If the score of the best-scoring rule drops below a predefined threshold, the learning cycle is ended. A learning cycle consists of the following steps:

1. All potentially relevant rules are discovered.
2. The rules that were discovered during the previous step are scored.
3. The rule with the highest score is selected and stored, and applied to the dummy corpus.

In this process, two parallel corpora are used: the gold corpus containing the correct headlines, and the dummy corpus which contains the compressions that are in progress. The learning cycle attempts to find rules that rewrite the dummy corpus to the gold corpus.

Relevant rules are discovered by traversing the dummy corpus trees, and applying each rule template to each tree node. Application of an rule template will generate the rule with conditions generated by the condition generators, except if a rule template is not relevant to a node. For instance, the application of a rule template with a condition generator that looks at sibling node attributes will not be useful for nodes that have no sibling.

The next step is to score all the rules that were discovered. This is done by applying each rule to a copy of each dummy corpus parse tree. Both the unmodified dummy corpus parse tree and the copy to which the rule was applied are compared to the reference sentence. This comparison is performed with the ROUGE-SU measure that is described in chapter 6.

That chapter also provides the rationale for choosing this ROUGE measure over the other ROUGE measures. If the application of the rule is found to give an improvement, the rule gets a point. If the application of the rule has a bad impact, a rule type-specific penalty is given. For instance, one could give a DeleteAction a penalty of -6, meaning that if a rule had a bad impact, 6 points are subtracted from its score. In the current system, the penalty points are specified per action type.

For example, let's consider the following reference sentence (R), the current dummy corpus sentence (D) and candidate sentences (*C1* and *C2*) that were generated through two possible rules:

```
R:  Spain wins European soccer championship
D:  Yesterday , Spain won the European soccer championship .
C1: Yesterday , Spain won the European soccer championship
C2: Yesterday , Spain won the European soccer .
```

C1 would be a good example of the frequent punctuation deletion rule, deleting the final period. The rule that was generated by second candidate is apparently not very useful (at least not in this context). The scoring function compares both *C1* and *C2* with R, and would in this case give scores of 0.53 and 0.28 respectively. Comparing the D with R gives a score of 0.45. So, in this case, the rule that generated *C1* will get a point, while the rule that generated *C2* will get the penalty that was assigned to deletion rules. Finally, the rule with the highest score is selected, and applied to the dummy corpus.

There is an additional problem that has to be dealt with: both the original sentences and the reference headlines vary in length. If the learning would proceed with all sentence/headline pairs the learning process would stagnate once some candidates have reached the length of the reference headline, since removing more words is likely to incur a penalty. There are two possible solutions to this problem:

- Set a headline length threshold that is used to remove a the candidate compression and reference headline when the candidate reaches that threshold.
- Remove a candidate compression and reference headline once the candidate compression has reached the length of the reference headline.

The first solution learns rules that are oriented towards trimming sentences to a headline with a certain length. For practical applications, this would be very useful. However, for this project this seems more problematic. First of all, if the threshold is set to a low number of words, the stagnation of learning resurfaces since it is a milder form of having no threshold. Secondly, the News2002-based test corpus consists of headlines of a varying

length, trimming headlines to a certain length would complicate evaluation, since we do not have a large set of reference sentences of one (predetermined) length. Obviously, comparing a longer compression with short reference sentences (or vise versa) will give bad scores.

The second solution has proven to be more useful. Since compressions that are 'finished' are removed, no stagnation occurs. Besides that, the rules that are learnt are not biased to a predefined trimming length, and can be used to evaluate with a test corpus that has a varying length of reference headlines. Therefor, candidate/reference pairs are removed once the length of the candidate compression has become equal to or less than the length of the reference headline.

While this concludes the description of the learning process, there are some interesting implementation details, such as the optimization of the learning cycle. These implementation details are described in chapter 7.

5.6 Trimming

After the generation of a rule set, trimming sentences is fairly simple. Each rule is applied in the order they were learnt to the parse tree of a sentence, until the requested length is attained. Since the system, like the hedge trimmer, does not work with probabilities, there is no optimal headline length. The user of a rule-based system asks the system to generate headlines of the length that is wished.

Chapter 6

Methodology for evaluation

During this project, evaluation is useful at two stages: for evaluating the headline generation method proposed in this thesis and other headline generation methods. Additionally, an evaluation method is required to guide the learning process. Traditionally, human judges were used to judge the quality of compression, focusing on aspects like grammaticality and importance of compressions. The most important downside to human judgement of compressions is that it is (time-)expensive. Additionally, human judges are not practically usable for the learning process.

Another methodology used is the BLEU system for automatic evaluation of machine translation[9]. BLEU uses a modified n-gram precision measure. If we see compression as a form of translation from longer sentences to shorter sentences, such a system can be used to compare compressed sentences to correct reference compressions.

The newer ROUGE system can be seen as as an extension to BLEU, oriented at automatic evaluation of summaries. ROUGE seems to be a very good candidate for evaluating headline generation, and for providing metrics to guide the learning process. In this chapter, a description of ROUGE will be provided, followed by a discussion of which ROUGE measures seem to be appropriate for learning tree compressions.

6.1 ROUGE

6.1.1 Introduction

ROUGE[1] is a system for measuring the quality of summaries by comparing it to a correct summary created by humans[1]. ROUGE provides four different measures, namely ROUGE-N, ROUGE-L, ROUGE-W, ROUGE-S and ROUGE-SU. The following sections will describe these measures, with an emphasis on how they can be used to compare a proposed compression

[1]Recall-Oriented Understudy for Gisting Evaluation.

to a correct compression. For a discussion of the ROUGE measures for comparing multiple-sentence summaries, refer to [1].

6.1.2 ROUGE-N

ROUGE-N measures co-occurances of n-grams. The ROUGE-N score can be calculated as:

$$ROUGE - N = \frac{\sum_{gram_n \in S} Count_{match}(gram_n)}{\sum_{gram_n \in S} Count(gram_n)} \tag{6.1}$$

Where S is the reference sentence, n is the length of the n-gram, $Count(gram_n)$ the number of n-grams co-occurring in the candidate and the reference sentence. Since the denominator is the total number of n-grams occurring in the reference sentence, ROUGE-N measures the n-gram recall.

Let's look at one short example of a reference compression R and a candidate compression C:

```
R: The cat is on the mat.
C: The cat on the mat.
```

In this example, the ROUGE-1 score is $\frac{5}{6}$, the ROUGE-2 score is $\frac{3}{5}$.

6.1.3 ROUGE-L

The ROUGE-L measure is based on the longest common subsequence (LCS) of a candidate sentence and the reference sentence. A subsequence Y of a sequence X is a sequence that occurs in X in incremental order. The LCS is the longest subsequence shared by two sequences. For instance, in the two sequences

```
ABCD
ABDE
```

the LCS is *ABD*. A sentence can be viewed as a sequence of words, over which the LCS can be calculated. A larger LCS could indicate that two sentences are more similar. Of course, a generated sentence could contain the reference sentence with a lot of additional noise. On the other hand, a compression could be very short but an exact substring[2]. The precision and recall measures are well fit to capture these potential situations:

$$R_{lcs} = \frac{LCS(X, Y)}{m} \tag{6.2}$$

[2]In contrast to a subsequence, a substring does not allow for gaps.

$$P_{lcs} = \frac{LCS(X,Y)}{n} \tag{6.3}$$

Where X is the reference compression with length m, and Y is the candidate compression with length n. The F-measure can then be used as a good measure to combine precision and recall:

$$F_{lcs} = \frac{(1+\beta^2)R_{lcs}P_{lcs}}{R_{lcs} + \beta^2 P_{lcs}} \tag{6.4}$$

where β is a constant to indicate the importance of recall over precision. F_{lcs} is defined to be ROUGE-L.

An advantage of ROUGE-L (over for example, ROUGE-N) is that besides capturing consecutive matches, it also accounts for non-consecutive in-order matches. This helps giving preference to candidates with a word-order that is comparable to the reference compression. For example, consider the following reference R and candidates *C1* and *C2*:

```
R: the cat was on the mat
C1: the cat is on the mat
C2: the mat is on the cat
```

Both candidate sentences have the same ROUGE-2 score ($\frac{3}{5}$), while in fact *C1* has a better word order than *C2*. ROUGE-L reflects this, because $LCS(R, C1) > LCS(R, C2)$.

A problem with LCS is that it does not take multiple LCSes, or shorter alternative subsequences into account. For example, in *C2* the LCS is "the on the". The sentence "the flea is on the cat" has the same LCS as *C2*, but since *C2* shares more meaning with R, it would be preferable to give it a higher score.

6.1.4 ROUGE-W

While ROUGE-L has good properties, it does not take the distances between the elements of the longest common subsequence into account. For example, in the reference sequence A and the candidates *C1* and *C2*

```
R:  A B C D E F G
C1: A B C D H I J
C2: A H B I C J D
```

C1 and *C2* share the same LCS with R, namely *A B C D*. But *C1* should be preferable to *C2*, because it has consecutive matches. ROUGE-W solves this deficiency by using a weighted variant of LCS (WLCS). The LCS of two sequences can be determined with a dynamic programming algorithm that uses a two-dimensional table. WLCS keeps an additional table that keeps

track of the number of consecutive matches. Rather than incrementing the LCS length with one when a new match is found, the length of the match is incremented with $f(k)$ where f is a weighting function, and k is the number of consecutive matches. This function must have the property that it gives consecutive matches get a higher score than non-consecutive matches. To be able to normalize the ROUGE-W score, a function with a close form inverse function is preferred. For instance, polynomial functions of the form $f(k) = k^n$ qualify, having a close form inverse function of the form $f^{-1} = k^{\frac{1}{n}}$.

Analogous to ROUGE-L, the recall, precision, and F-measure can be calculated:

$$R_{wlcs} = f^{-1}\left(\frac{WLCS(X,Y)}{f(m)}\right) \tag{6.5}$$

$$P_{wlcs} = f^{-1}\left(\frac{WLCS(X,Y)}{f(n)}\right) \tag{6.6}$$

$$F_{wlcs} = \frac{(1+\beta^2)R_{wlcs}P_{wlcs}}{R_{wlcs} + \beta^2 P_{wlcs}} \tag{6.7}$$

where $WLC(X,Y)$ is the weighted LCS of the reference compression X with length m and candidate compression Y with length n. Additionally, f^{-1} is the inverse version of f used in WLCS.

6.1.5 ROUGE-S

ROUGE-S measures the coverage of skip-bigrams by the candidate sentence. A skip-bigram is any pair of words in sentence order, with arbitrary large gaps. For instance, consider the following reference sentence *R*, and candidate sentences *C1*, *C2*, and *C3*:

```
R:  police killed the gunman
C1: police kill the gunman
C2: the gunman kill police
C3: the gunman police killed
```

Each of these four word sentences has $\binom{4}{2} = 6$ skip-bigrams. For example, R has the skip-bigrams "police killed" "police the", "police gunman", "killed the", "killed gunman", and "the gunman". The number of skip-bigram matches can of the reference compression and the candidate compression can then be counted. Analogous to ROUGE-L and ROUGE-W, the recall, precision, and F-measure can be calculated:

$$R_{skip2} = \frac{SKIP2(X,Y)}{\binom{m}{2}} \tag{6.8}$$

$$P_{skip2} = \frac{SKIP2(X,Y)}{\binom{n}{2}} \tag{6.9}$$

$$F_{skip2} = \frac{(1+\beta^2)R_{skip2}P_{skip2}}{R_{skip2} + \beta^2 P_{skip2}} \tag{6.10}$$

where $SKIP2(X,Y)$ is the number of skip-bigram matches between the reference compression X with length m and the candidate compression Y with length n. As expected, ROUGE-S is defined to be F_{skip2}.

With $\beta = 1$, the scores of the candidate are respectively 0.5, 0.167, and 0.333. So, *C1* is better than *C2* and *C3*, *C3* is better than *C2*. As we can see in this example, ROUGE-S solves one of the problems with both LCS-based measures - although *C2* and *C3* both have the same LCS, but *C3* has a better ROUGE-S score because it takes all bigrams that are in-order into account.

If the gap in a skip-bigram is unbounded, it may result in spurious skip-bigrams such as "the the". This can be solved to some extend by specifying the maximal size of a gap. For instance, a gap of 0 would result in normal bigrams, a gap of 4 would allow for four words to exist between a skip-bigram. Of course, the denominator should of the recall and precision should be changed to the maximum number of skip bigrams (as the combination as used above computes the number of skip bigrams when any gap is allowed).

6.1.6 ROUGE-SU

On potential problem with ROUGE-S is that it does not give any value to candidate compressions without matching skip-bigrams, even if many words are shared. For example the compression "gunman the killed police" fully matches on unigrams with the reference compression from the previous section, because it is just the inverse sentence, but it does not have any matching skip-bigrams.

ROUGE-SU still gives credit to such cases by also counting matching unigrams. The same effect can be achieved by left-padding the reference and candidate compressions by an additional marker word, which causes the marker plus every word to be a skip-bigram.

6.2 Using ROUGE as a scoring function for TBL

As described in section 5, a method to compare compressions generated by candidate rules with the reference headline is required. Such method requires three characteristics:

1. A method should take good word overlap into account. This avoids that rules that often delete important words or constituents get good scores.

2. A method should be sensitive to the word order. If two candidates have the same words, but one of the candidates has a word order that resembles the reference headline more closely, it should get a higher score.

3. A method should take the differences in length into account. A rule that removes unimportant words that do not occur in the reference sentence should get rewarded.

None of these characteristics require a method that compares the parse trees. It was decided to use a word-level measure, because comparing parse trees can give unnecessary complications. For example, the parse tree of short reference headlines often differs highly from the parse tree of the first sentence. Besides that, most transformations do not take the correctness of the resulting trees into account. It may be useful in the future to check the trees generated by transformations against a grammar. But, for now it was decided to use one of the ROUGE measures for compression comparison.

All the ROUGE measures have the third characteristic, since they embody the precision and recall, all measures are normalized over the length of a sentence or an aggregate thereof. With regard to this characteristic, all of the ROUGE measures are potential candidates.

ROUGE-N does confirm to the first characteristic as well, at least if N is small (preferably 1). If N is larger, both sentences may share many words, but if their order is different, there may not be many similar n-grams. Although ROUGE-N does conform to the first characteristic, for a small N, it only possesses the second characteristic weakly. For instance, ROUGE-2 gives higher scores if the order of words is comparable, but does not account for all non-consecutive in-order matches (for instance, consider the example given in section 6.1.3).

ROUGE-L and ROUGE-W have the second characteristic, they are very sensitive to word orders. However, as previously mentioned, it gives little or no points when the word order differs highly, but a compression still has many correct words. Additionally, it only takes the longest subsequence into account. As such, it is a bad candidate for scoring in the learning cycle, it is not sensitive to many changes that are improvements.

The remaining candidates, ROUGE-S, and ROUGE-SU have all required characteristics: more overlap in skip-bigrams is found when the word order is comparable. While ROUGE-S takes word overlap in account fairly well (like ROUGE-N where N is 2), it does not do this as well as ROUGE-1. If words happen to have different orders in both sentences, there may be little skip-bigram overlap. ROUGE-SU solves this, by practically including unigrams in the ROUGE-S measure. Since ROUGE-SU is sensitive to both word overlap and word order, it seems to be the most fit candidate for sentence comparison during learning cycles.

6.3 Using ROUGE for evaluation

ROUGE was also used to perform automatic evaluation of the headlines generated from the test set. Ideally, the evaluation measure should have the same characteristics as those outlined in the previous section. However, it would not be fair to use exactly the same measure, since the generated rules may have a bias to generating headlines that have a high ROUGE-SU score.

To measure the word order, the ROUGE-L measure is used. However, it would be useful to capture unigram overlap as well, since the trimmer may have kept the correct words, even if their order is mostly different. For this reason, ROUGE-L is accompanied by the unigram precision of word roots in the machine evaluation.

Chapter 7

Implementation details

7.1 Speeding up learning

If the learning algorithm is literally implemented as described in section 5.5, a lot of unnecessary work is done. Since most rules only affect a (small) subset of all sentences, it is not necessary to generate rules over all trees during every learning cycle.

To avoid these inefficiencies, various optimizations were used for the learning cycle, that are described in the pseudocode shown in program listing 2. During each learning cycle a list of indexes of trees that were changed during the previous cycle should be available. To bootstrap the learning, the set of changed indexes is initialized to the set of all tree indexes, and the rule/index score map is empty.

This algorithm is shares some similarities with the TBL algorithm described by Ngai and Florian[10], in that it does not regenerate the list of rules during each cycle, but keeps track of the rules and trees that are potentially affected by a cycle, keeping the need to regenerate and rescore rules to a minimum.

7.2 Handling of Alpino parse trees

As described in chapter 5 the system uses the Alpino XML format for parse trees. While common class libraries, such as the Java class library, provide functionality to represent XML files as DOM[1] trees, DOM trees may not be the ideal tree representation. The primary problem with DOM is its memory overhead, which should be avoided when potentially loading thousands of trees. For this reason, the system reads Alpino trees using the DOM API, but instantly converts them to an internal tree representation. This internal tree format stores every attribute found for nodes in the Alpino trees as attributes in a map that is associated with a tree node, except for

[1] Document Object Model

the primary attributes that are used in every node: the begin/end positions and the node ID. Except for references to children, nodes should also have a reference to their parent, to simplify the writing of conditions.

A task that often needs to be performed is to extract the actual sentence that is represented by a dependency tree, or the roots of the words that form the sentence. It's not enough to gather nodes in a left to right order, since the tree order may not correspond to the actual sentence word order. Fortunately, all nodes have indications of their positions through their begin and end attributes. So, retrieve a sentence, we first select all nodes that have a *word* attribute, and store them in a list. Then we sort this list on the *begin* attribute. The leaf nodes are then ordered in the sentence word order, so all words can then be extracted from the nodes and put in an array that represents the sentence words. The same procedure can be used for word root, by using the *root* attribute in place of the *word* attribute.

Listing 2 Learning cycle algorithm pseudo-code

```
// 'rules' and 'changedIndexes are carried over from the
// previous learning cycle.
for (i in changedTreeIndexes) {
  treeRules = gatherTreeRules(dummyCorpus[i]);
  rules[rule][i] = 0;
}

// Find the scores of rules for trees at changedTreeIndexes.
scoreRules(goldCorpus, dummyCorpus, rules, changedTreeIndexes);

// Find the rule with the highest summed score, and store it.
bestRule = findBestRule(rules);
results += bestRule;

// Apply the rule. We only need to apply it to trees that
// we know the rule applies to.
applyRule(dummyCorpus, bestRule, rules[bestRule].keys());

// Store the changed indices for the next cycle.
changedTreeIndexes = rules[bestRule].keys();


// Purge sentences that have a sufficient length. We
// can do this by discarding their indexes from the
// changedTreeIndexes set: only trees affected by this
// cycle can reach a sufficient length during this cycle,
// and during the next step we will purge all rule score
// entries for the changedIndexes. As a result these trees
// will never be analyzed again.
changedTreeIndexes = purgeSentences(goldCorpus, dummyCorpus,
  changedTreeIndexes);

// Purge the rule scores for the changed indexes, a rule may
// not affect one of the changed trees during the next cycle.
purgeScores(rules, changedIndexes);
```

Chapter 8

Results

The system described in this thesis was evaluated with the News2002 corpus that is described in chapter 4. This corpus consists of 7233 headline/sentence pairs that were extracted from the Dutch AD, NRC, Parool, Trouw, and Volkskrant newspapers from 2002. The first 90% of the corpus was used as training material, the remaining 10% as a test set. During the development a subset of the first 90% was used for training and testing the system.

For the evaluation of the system, the system was trained on two sets of rule templates: the rule templates that mirror the rules used in the hedge trimmer (HT-Like), and the same set with the addition of some extra rule templates as described in chapter 5 (Extended).

Two evaluations were performed: both machine evaluation and evaluation by human judges.

8.1 Machine evaluation

For the machine evaluation of the trimmer with these rule templates, the ROUGE-L measure is used, as motivated in section 6.3. Additionally, the (unigram) word root precisions were also calculated.

This evaluation was performed by applying the rules to the test corpus. Headline candidates that have reached the same length as the reference sentence are excluded from further trimming (otherwise it would be difficult to evaluate the compressions, as set forth in 5.5). The trimming concludes when all rules have been applied or when all sentences have been trimmed to the length of their reference sentences[1]

The type of evaluation performed here may seem a bit problematic, because the target length is hinted to the trimmer. Other trimmers, such as the hedge-trimmer have been evaluated with a similar task, such as the 10-word summarization task of the DUC2003 conference[2]. The general idea is

[1] In practice, 'rule starvation' occurs earlier.

[2] http://duc.nist.gov/duc2003/tasks.html

to test the quality of the headline when it is trimmed to a prespecified length (e.g. 10-words for this DUC2003 task). But since no such task is available for Dutch, the approach described above simulates this with varying lengths. This was one of the motivations to perform a human evaluation as well.

The result of this evaluation are shown in table 8.1. The scores for the uncompressed headline are also shown in the *Sentence* column. Since there is no good baseline yet to compare this system with, this shows the improvement over the untrimmed sentence.

Table 8.1: Average ROUGE-L score and unigram precision for the uncompressed sentences, the hedge trimmer-like trimmer, and the extended trimmer.

	Sentence	HT-like	Extended
ROUGE-L	0.324 (0.183)	0.434 (0.288)	0.439 (0.290)
Unigram precision	0.272 (0.158)	0.505 (0.310)	0.511 (0.309)

As the results show, a clear improvement over the uncompressed sentence is shown. There only seem to be very minor ROUGE-L score and unigram precision differences between the HT-like and Extended trimmers, with a slight edge for the extended trimmer.

Given the circumstances, the scores seem fair, since there are three factors that push ROUGE-L scores down:

1. Often, the trimmer will come up with slightly different, or even completely different headlines, that are also correct or only slightly worse.

2. ROUGE-L only takes the longest subsequence into account. If the original sentence has a different word order than the reference headline, often the generated headline is also fine, it just had a (partly) different word order.

3. After all rules are applied, the generated headlines are sometimes still longer than the reference headlines. There are many short headlines in the training and test sets, and it is hard to get to such short lengths without making overly aggressive rules.

Given the unigram precisions, the first and third factors are likely causes for ROUGE-L score degradation, besides incorrect trimming. The third factor could be confirmed by looking at the number of headlines that were still longer than their reference candidate during the machine evaluation. After application of the HT-like and Extended rules, 44.5% and 44% of the headlines were longer than the reference headlines respectively.

Given these possible distortions of the results, it was also appropriate to conduct an evaluation with human judges.

8.2 Human evaluation

For the human evaluation, 50 sentences were chosen randomly from the test corpus. These sentences were trimmed to a target length of 8 words, meaning that headlines that have reached this length are removed from further trimming. This length was chosen because it seems to be a fair length for headlines, and usually leads to a compression of around 50% of the original sentence (depending on the sentences that the computer randomly selected). Comparable compressions rates are used for the evaluation of other systems.

The two judges were given the original sentence and the headlines generated by the HT-like and Extended rules, and were asked to rate each sentence on a scale from 1 (bad) to 5 (excellent) for grammaticality and relevance. Table 8.2 shows the average scores and their standard deviations retrieved from this evaluation.

Table 8.2: Human judgements of the hedge trimmer-like and extended trimmers.

	HT-like	Extended
Grammaticality	3.85 (1.03)	3.75 (1.05)
Relevance	3.82 (1.13)	3.76 (1.14)

Somewhat surprisingly, the rules generated with the extended set of templates perform worse than the rules generated with the templates that mirror the hedge trimmer. While the differences are small, and it is hard to distinguish consistent differences between the HL-Like and extended headlines, the Extended headlines seem to be a bit more aggressive in deleting nouns and proper nouns. The two categories of deletion templates in the Extended rule templates are generally less specific than the HL-like templates. Though, the Extended set may perform better for generating very short (i.e. two or three word) headlines. This may also give the edge to the Extended set in the ROUGE-L evaluation, since the machine evaluation includes many short headlines as well.

These results cannot be compared directly to the results of the human evaluation of the hedge trimmer (as summarized in section 2.2), since the evaluation of the hedge trimmer does not separate grammaticality and relevance, and ideally the systems should be compared by the same judges. Still, it is shows that this system performs about as well as the hedge trimmer. The average of the grammaticality and the relevance of the HT-like rule set is 3.84, compared to 3.72 for the hedge trimmer.

Compared to the noisy-channel model, as described in section 2.1, this system generates headlines that are less grammatical, but it is better at extracting important fragments. This may not be surprising: the noisy-channel

model uses PCFG probabilities to weight every compression. As a result it is very able at producing grammatical sentences. However, preference to grammaticality may prevent removal of unimportant phrases. Additionally, headlines are often not strictly grammatical.

Given the difficulties inherent to the data set, compared to the data sets that were used for training and testing most English trimmers[3], it seems that a trimmer based on transformation-based learning is well able to capture rules that are comparable to those of the hedge trimmer. Additionally, due to the generic nature of rule templates, it's easier to capture more specific rules. The fact that Alpino provides information about dependency relations helps with generating very specific rules as well.

8.3 Examples of trimming problems

In this section, a few typical problems that are found during trimming are discussed.

8.3.1 Verb inflection

One typical problem is the ungrammaticality caused by trimming where past participles are involved. For instance, consider the following sentence and the proposed headline:

```
S: De Amerikaanse wielrenner Tyler Hamilton heeft zondag bij
   een training voor de GP Eddy Merckx een sleutelbeen
   gebroken .
   (The American cyclist Tyler Hamilton has Sunday at a
   training for the GP Eddy Merckx a clavicle broken .)
H: wielrenner Tyler Hamilton sleutelbeen gebroken
   (cyclist Tyler Hamilton clavicle broken)
```

Since the auxiliary verb 'heeft' ('have') is removed, the inflection of the verb 'gebroken' ('broken') should change and take the place of the auxiliary verb.

A rule type that performs such a modification was written and tested, and it generally seems to work. The problem is that it is very language-specific and sacrifices the generality of the system, and would practically be closer to manual rule creation than machine learning. Additionally, it turned out to require very specific tuning - a rule that removes all verbs with the root 'heb' ('have') normally has a higher score, so a rule that modifies verbs in this manner requires additional boosting.

[3] As described in chapter 4, the headline words do not necessarily have the same order as the uncompressed sentence in the News2002 corpus.

8.3.2 Non-salient sentences

As previous research has shown, usually the first sentence contains the majority of useful headline words. However, sometimes the first sentence is just a lead in. Consider for example the sentence "Engels spreken ze nauwelijks - en anders zouden ze nog altijd weinig zeggen: Jonsi , Kjarri , Oggi en Orri van het schuwe, IJslandse Sigur Rós." (literal: "English speak they barely - and otherwise should they even always little say: Jonsi, Kjarri, Oggi and Orri of the shy, Icelandic Sigur Rós"). The first sentence is a characterization of the band members of Sigur Rós, as an introduction to an article. It's difficult to extract relevant words from such a sentence. A possible headline based on this sentence could just be "Sigur Rós". However, if the system is asked to create a headline of around eight words, it's hard to come up with a fitting headline. With both rulesets, the following headline is generated: "Jonsi Kjarri Oggi en Orri schuwe IJslandse Sigur Rós" (literal: "Jonsi Kjarri Oggi and Orri shy Icelandic Sigur Rós"). While proper names are usually important words, here it creates a meaningless headline.

8.3.3 Insufficient trimming

Sometimes, the trimmer does sensibly cut off some material, with the result that the remaining sentence is ungrammatical and incomplete, where additional trimming would have corrected this ungrammaticality. For instance, consider the following example of a sentence and a proposed headline:

```
S: Voorzitter en penningmeester Gerrit Bloemink van FC Utrecht
   stapt op per 1 oktober.
   (Chairman and treasurer Gerrit Bloemink of FC Utrecht
   quits starting 1 October.)
H: en penningmeester Gerrit Bloemink stapt op per
   (and treasurer Gerrit Bloemink quits starting)
```

(Note that 'stap op' is an inflection of the verb 'opstappen', which means 'to leave' or 'to quit', for convenience this was translated with one word.) The generated headline is flawed, the noun 'Voorzitter' was deleted, leaving only partial information about 'Gerrit Bloemink'. 'FC Utrecht'[4], is also a crucial name in this sentence, is removed too. A more useful deletion that took place was the removal of the time expression '1 oktober'. However, the deletion kept the dangling word 'per'. Subsequent deletions would probably have removed this word, but the threshold for the trimming was reached, leaving the headline as-is. In contrast to e.g. the noisy channel model, this system does have no way to determine whether another deletion is useful. This is an inherent weakness that systems that can verify or estimate the grammatical correctness of a sentence do not have.

[4]FC Utrecht is a Dutch soccer club.

One possibility to work around this problem is to check headline candidates that have a length that is near the target length using a parser. The candidate that proves to be the most grammatical could then be selected as the candidate headline, even if it is longer or shorter than the reference length.

Chapter 9

Conclusions

9.1 Overview of the results

In this thesis I have attempted to apply transformation-based learning to the task of sentence compression and headline generation in particular. Analogous to transformation-based learning in other tasks, such as part-of-speech tagging, the TBL system that was developed generates understandable rules that reflect linguistic insights. A clear advantage over most other rule-based systems is that this does not require the manual work that is normally involved in writing rules that are bound to a specific language and parser.

The system developed for this thesis can relatively easily be modified by adding a reader class for the specific parse tree format that a parser produces, and modifying rule templates to use the attributes that are assigned to nodes by that specific parser. The learning cycle and condition/action classes are independent of the parser or language that is used. As such, the system can compete with systems that rely on statistical techniques. Arguably, even less parser-specific information is required, because this system does not require PCFG probabilities.

Of course, a system for transformation-based learning still needs to be seeded with rule templates to learn actual rules. Since the parse trees that are produced by the Alpino parser and grammar are quite different from those produced by the BBN parser that is used by the hedge trimmer, those rules can not be used directly (of course, this would also sacrifice the generic nature of a TBL system). Still, the linguistic motivations for the hedge trimmer are sound, so I have tried to capture the rules in more abstract rule templates. As the results indicate, this generates a rule set that is well-equipped to perform trimming tasks.

An additional problem that has been dealt with was the lack of training material for the generation of Dutch headlines. In this thesis I have described a method for automatic extraction of training material from a Dutch newspaper corpus. While the training data is relatively fuzzy, since

the word order of the headlines does not correspond with the word order of the first sentence of an article, and there is an overwhelming amount of very short headlines, the corpus prove to be useful for training and testing the system. In the future, it would be interesting to see how the performance of the system increases when less fuzzy material is available and used.

9.2 Possible improvements

In retrospect, it's probably worth making deletion actions less destructive. Currently, the system removes nodes in the candidate headlines during the learning and trimming phases, e.g. when a *DeleteAction* is applied. In this respect, the system differs from traditional TBL systems, that only apply transformations. A different approach that could be tried is seeing deletion as setting a on/off flag on a node. If a node is flagged as 'off', the subtree should not be used in the candidate headline. This would allow for rules that can re-enable nodes in more specific contexts. For example, a rule could disable words with the root 'heb' ('have'), but re-enable it through rules that have a more specific context. Of course, if such a deletion policy would be implemented, it needs to be decided what should be done with actions that change the structure of the tree (e.g. for *ReplaceParentAction* instances).

Another problem that would still be interesting to solve is verb inflection. Especially the placement and inflection of past participles needs to be corrected (as discussed in section 8.3.1). While it is relatively easy to create a language and parse tree-specific rule, this is not very elegant in the context of a TBL system, and a more generic approach would be appropriate.

Finally, the grammaticality of headlines could be improved by adding verification. If the parser that is used to generate sentence parse trees can assign probabilities to parses, the probabilities of headlines with lengths around the headline length threshold could be estimated during trimming. Consequently, the most probable headline could be chosen. It seems less useful to use such probabilities during the learning phase. Often it is not harmful to make a headline ungrammatical by applying a rule, if a subsequent rule also removes the remaining fragments that make a sentence ungrammatical. Application of probabilities during the trimming process can solve the problem described in section 8.3.3.

9.3 Final remarks

I sincerely hope that the system and methodology set forth in this thesis will prove to be useful. At the very least, it seems promising enough as a generalization of the hedge trimmer and an application of TBL for the task set forth in this thesis. Due to its generic design, the system described in

this thesis can easily be expanded for further research. Hopefully, this can provide further insights in the process of sentence compression.

Bibliography

[1] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In Marie-Francine Moens Stan Szpakow-icz, editor, *Text Summarization Branches Out: Proceedings of the ACL-04 Workshop*. Association for Computational Linguistics, 2004.

[2] Ryan Mcdonald. Discriminative sentence compression with soft syntactic constraints. In *In Proceedings of the 11th EACL*, 2006.

[3] Kevin Knight and Daniel Marcu. Summarization beyond sentence extraction: A probabilistic approach to sentence compression. *Artif. Intell.*, 139(1):91–107, 2002.

[4] John M Conroy, Judith D Schlesinger, Dianne P OLeary, and J Goldstein. Back to basics: Classy 2006. In *In Proceedings of the 2006 Document Understanding Conference (DUC 2006) at HLT/NAACL 2006*, 2006.

[5] Yuya Unno, Takashi Ninomiya, Yusuke Miyao, and Jun'ichi Tsujii. Trimming cfg parse trees for sentence compression using machine learning approaches. In *Proceedings of the COLING/ACL on Main conference poster sessions*, pages 850–857, Morristown, NJ, USA, 2006. Association for Computational Linguistics.

[6] B. Dorr, D. Zajic, and R. Schwartz. Hedge trimmer: A parse-and-trim approach to headline generation, 2003.

[7] Noam Chomsky. *Lectures on Government and Binding*. Foris, Dordrecht, 1981.

[8] Eric David Brill. *A corpus-based approach to language learning*. PhD thesis, University of Pennsylvania, Philadelphia, PA, USA, 1993.

[9] K. Papineni, S. Roukos, T. Ward, and W. Zhu. Bleu: a method for automatic evaluation of machine translation, 2001.

[10] Grace Ngai and Radu Florian. Transformation-based learning in the fast lane. In *NAACL '01: Second meeting of the North American Chap-*

ter of the Association for Computational Linguistics on Language technologies 2001, pages 1–8, Morristown, NJ, USA, 2001. Association for Computational Linguistics.