Institute for Natural Language Processing

University of Stuttgart
Pfaffenwaldring 5b
D–70569 Stuttgart

Diploma Thesis No. 3568

Graphical Error Mining For Linguistic Annotated Corpora

Gregor Thiele

Course of Study:	Computer Science
Examiner:	Prof. Dr. Jonas Kuhn
Supervisor:	Dipl.-Ling. Wolfgang Seeker
Commenced:	April 19, 2013
Completed:	September 27, 2013
CR-Classification:	D.1.5 H.5.2 J.5

Abstract

Corpora contain linguistically annotated data. Producing these annotations is a complex process that easily leads to inconsistencies within the annotation. Since corpora are used to evaluate automatic language processing systems the evaluation may suffer when there are too many errors within the data.

This thesis focuses on finding erroneous annotations within corpora. To detect sequence annotation errors within part-of-speech tags we implemented the algorithm introduced by Dickinson and Meurers (2003). Additionally for structured annotations we choose the approach shown in Boyd et al. (2008) that targets inconsistency within dependency structures.

We designed and built a graphical user interface (GUI) that is easy to handle and user-friendly. Implementing state-of-the-art algorithms for error detection with an user-friendly interface increase the operation domain because the algorithms can be used by a wider audience without deeper knowledge of computers. It provides even non-expert users with the capability to find inconsistent pos tags and dependency structures within a corpus. We evaluate the system using the German TIGER corpus and the English Penn Treebank. For the TIGER corpus we also perform a manual evaluation where we sample 115 6-grams and check manually if these contain errors. We find that 94.96% are erroneous and it is easy to decide the correct tag as a human. For 4.20% we can say that these are errors but determining the correct tag is very to difficult. In total we detect errors with a precision of 99.16%. Only one case (0.84%) is not caused by inconsistency but constitutes genuine ambiguity.

Contents

List of Figures

List of Listings

List of Tables

List of Algorithms

1 Introduction and Motivation

Processing language requires a format to store text with useful information needed by linguists. These datasets are called corpus. They contain structured language data of texts with linguistic mark-up. For example words may have one or more of the following information assigned: part-of-speech, lemma, morphology,... Corpora are important for evaluating algorithms and methods. The evaluation quality depends on the given corpus. Therefore it is preferable that the corpus data an algorithm is evaluated against has only few inconsistencies. In this thesis we will work with algorithms that may help detecting these errors to assists human annotators in finding and correcting them.

We distinguish between gold- and system-annotated data. Gold annotated data is produced by humans, or to be more general the last decision which tag is assigned is done by human annotators. An example for a gold standard corpus is the German TIGER Corpus (Brants et al., 2004) and for English the Penn Treebank (Marcus et al., 1993) or the British National Corpus (BNC) (Geoffrey et al., 1994). To evaluate our implementation we used the TIGER Corpus and Penn Treebank.

Annotating a corpus involves different human annotators. The assumption that all annotators assign every time the correct part-of-speech tag hold for simple constructions where the annotation scheme is clear. But language is ambiguous and for one wordform the annotation rules may allow more than one tag for the same context. Additionally, inconsistency may be introduced by human errors.

If annotation is done by humans the decision for the correct tag may be less complicated. While reading a sentence they subconsciously use information about the context and the meaning of the sentence to determine the correct tag. Besides that they may have specific language domain knowledge about the corpus. Nevertheless there is no guarantee that all annotators pick the same tag in the same context.

Ratnaparkhi (1996) runs a consistency check on the Wall Street Journal (WSJ) looking at words and how they were tagged by different annotators. He shows that human annotators may introduce inconsistency because there is no guarantee that once the annotator corrects something this correction is applied every time it appears in the corpus. The resulting corpus may contain different annotations for one specific word usage.

An example of how difficult a tagging decision can be shows Example 1.1. The word *Press* could be tagged as NN (noun, singular or mass) or NNP (noun, proper singular).

(1.1)	*The*	*White*	*House*	***Press***	*Secretary*	*.*
	DT	NNP	NNP	**NN**	NN	.
	DT	NNP	NNP	**NNP**	NN	.

Another example can be found in Example 1.2. The sentence occurs twice in the Penn Treebank. One time *married* is tagged VBN (verb, past participle) and in the other case JJ (adjective).

(1.2) The margin of error for subgroups – for example, **married** women with children at home – would be larger.

Example 1.3 shows one sentence (of the German TIGER Corpus) where *Braque*, the surname of "Georges Braque" (french painter and developer of the art style cubism), is tagged as NE (in total we found 12 sentences where *Braque* is tagged as NE). But there is one sentence (Example 1.4) where *Braque* has misleadingly been tagged NN.

(1.3) 1 of 12 occurences where Braque is tagged as NE (proper noun)

- 1908 malt **Braque**, noch ganz unter dem Einfluß Cézannes, im südfranzösischen L'Estaque die ersten Bilder mit kubisch vereinfachten Formen und einer auf Grün, Braun und Ocker reduzierten Palette.

(1.4) Braque misleadingly tagged as NN (common noun)

- Der Kubismus, für Picasso letztlich ein Intermezzo, bleibt für **Braque** die wegweisende Entdeckung, die es ermöglicht, "nicht eine anekdotische Tatsache wiederzugeben, sondern eine malerische Tatsache zu geben".

All these words have something in common: they have more than one tag assignment in the corpus. Words with more than one tag may occur because of the following two possibilities as Dickinson and Meurers (2003) pointed out: First there is language ambiguity allowing multiple tags for one wordform which will lead to variation. Secondly there is an erroneous tagging which means that one word has at least two different tag assignments across comparable corpus instances. But it is not sufficient just to look at a single word when determining the correct tag as we show this in Example 1.5 and Example 1.6. The word *vage* has two different pos tags ADJA and ADJD. Both tags are proper for the given context. The variation is caused by genuine language ambiguity.

To determine if the tagging was erroneous we need comparable instances, which means the word can be found in at least two or more sentences **within the same word context**.

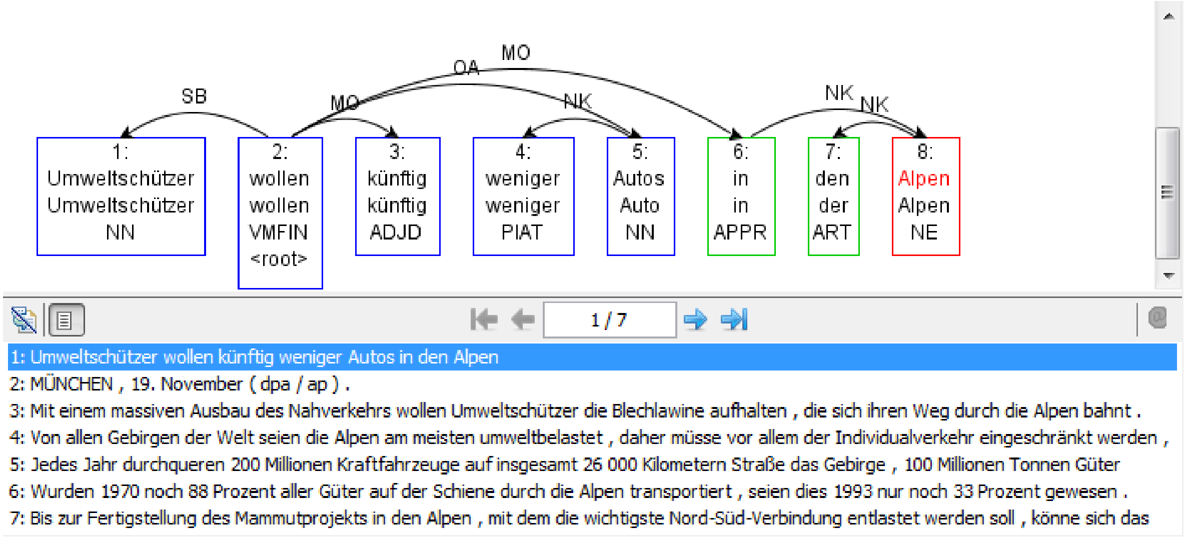
(1.5) 1 of 3 occurences vage tagged as ADJA:

- Zwar gibt es **vage** Pläne, die Kopenhagener Anlage mit der Kockums-Werft im schwedischen Malmö zu fusionieren.

(1.6) 1 of 7 vage tagged as ADJD:

- Die Ankündigung für das California Institute for European-American Relations liest sich ehrenwert und etwas **vage**, und so klingt es auch aus den Mündern der Beteiligten.

In Error mining, we are interested in finding the second case, the incorrectly assigned tags for one word within the same context. But detecting these annotation errors is challenging and for huge datasets this can only be done assisted by a computer. An algorithm to automatically detect annotation errors was introduced by Dickinson and Meurers (2003) for part-of-speech tags (henceforth pos tags) and was adapted later to dependency structures as well (proposed in Boyd et al. (2008)). The authors use variation within n-grams. N-Grams are used to map the same contexts together which is important since we want to compare strings only when they occur in the same context. A uni-gram is called a *variation nucleus*, if it has at least two different tag assignments. A list containing all nuclei in the entire corpus is generated during the algorithm initialization. This list is used to discover the longest n-grams for each nucleus. Bi-grams, or larger n-grams may contain more than just one nucleus. The n-grams generated with at least one nucleus are expected to be possible error candidates. Filter heuristics are further used to increase the precision of the algorithm.



1: Umweltschützer wollen künftig weniger Autos in den Alpen
2: MÜNCHEN , 19. November (dpa / ap) .
3: Mit einem massiven Ausbau des Nahverkehrs wollen Umweltschützer die Blechlawine aufhalten , die sich ihren Weg durch die Alpen bahnt .
4: Von allen Gebirgen der Welt seien die Alpen am meisten umweltbelastet , daher müsse vor allem der Individualverkehr eingeschränkt werden ,
5: Jedes Jahr durchqueren 200 Millionen Kraftfahrzeuge auf insgesamt 26 000 Kilometern Straße das Gebirge , 100 Millionen Tonnen Güter
6: Wurden 1970 noch 88 Prozent aller Güter auf der Schiene durch die Alpen transportiert , seien dies 1993 nur noch 33 Prozent gewesen .
7: Bis zur Fertigstellung des Mammutprojekts in den Alpen , mit dem die wichtigste Nord-Süd-Verbindung entlastet werden soll , könne sich das

Figure 1.1: Error mining result presentation for the 3-gram <in den Alpen>. N-gram coloured green, variation nucleus is marked red.

Most of the tools for finding errors are script based (pearl, python,...) and have no graphical user interface. Using them requires some expertise since there must be knowledge of the particular script language to execute them. Likewise the results are often unclear to non-expert users, because the script's output is often only text with numbers and words which is difficult for humans to read and interpret.

This thesis focuses on finding differently tagged words in a treebank for part-of-speech tags and for dependency structures, using the algorithm proposed in (Dickinson, 2005, p. 33-39). Secondly, we build a clear graphical user interface (GUI) to help the user error-mine his corpus, with only a few mouse interactions. We build the program "office" look-a-like so that the menu structure is clear and handling can be adopted from known programs. Besides the algorithm implementation itself we integrated both algorithms into the ICARUS search engine introduced by Gärtner et al. (2013) to allow an easy usage and clear structured result presentation of the prospect error candidates.

The developed error-mining plug-in lets the user interact graphically with the error mining algorithms without the need of a specific script language and produces a graphical output of the data. We implemented an n-gram structure that contains information about all sentences where a given n-gram occurs and stores information about all nuclei within. This is essential for the result highlighting we do later on. An example screenshot of the program is shown in Figure 1.1.

We developed a query mechanism that uses lookup tables to allow the user to redefine/overwrite existing tags during the search process. Modifying tags allows more general results e.g. by mapping all verb pos-tags (VVFIN, VVIMP, VVINF,...) to a simple *verb-tag*. Such a mapping is useful to detect words which were tagged as verbs but cannot be verbs since all words which are tagged as verbs can be found within the specified *verb-tag* class.

ICARUS is a fully portable application (only requirement installed Java 1.7[1] or higher) and is an easy way to provide non-expert users with a stable environment to work with the error mining plug-in. This thesis shows an easy way to detect erroneous tagged items in corpora within just a few mouse clicks.

The evaluation was done using the implemented algorithms with different parameters, for part-of-speech and dependency structure error mining. For German we used the TIGER Corpus and for English the training part of the Penn Treebank. We modified tags during the search to generalize or restrict the results to one phenomenon. Last we did manual evaluation where we sampled a fixed number of n-grams the algorithm delivered and manually determined if these are *true* errors, language ambiguity, or too difficult to determine the correct tag. We find out that our algorithm works with a precision of 99.16%.

The thesis is organized as follows: Chapter 2 introduces Dickinson's error detection algorithm for part-of-speech tags proposed in (Dickinson, 2005, p. 33-34). It also presents the error detection algorithm for dependency structures introduced by Boyd et al. (2008) and describes the implementation of both algorithms. We also describe the new features e.g. the tag query editor or the number replacement option. In Chapter 3, we evaluate the algorithm using the German TIGER Corpus and the English Penn Treebank. The evaluation was done using part-of-speech and dependency structure error mining. For the German TIGER Corpus we also carried out a manual evaluation. In Chapter 4, we discuss other work and how it relates to our own. We conclude with Chapter 5 including a short outlook on future work.

[1] www.java.com/de/download/

2 Graphical Error Mining

As we mentioned before in the introduction, corpora may contain inconsistencies and finding these is difficult. One possibility is to hire a few annotators and let them validate a corpus and then decide the correct tag by majority decision. But this would be a very expensive process. We need automated systems that assist us in finding errors or at least deliver us with a set of possible error candidates that can be reviewed manually.

At this point we want to introduce our automated error mining pipeline. The Figure 2.1 shows a simplified overview of the error mining process as it is implemented. The user chooses between the search-modes for error mining (part-of-speech / dependency structures) then specifies the target corpus. Secondly, further constraints can be set using the query editor (see Figure 2.5). These queries are used to redefine tags during the execution but do not change the treebank data. Finally search parameters can be set. When all desired settings are made the search can be executed.

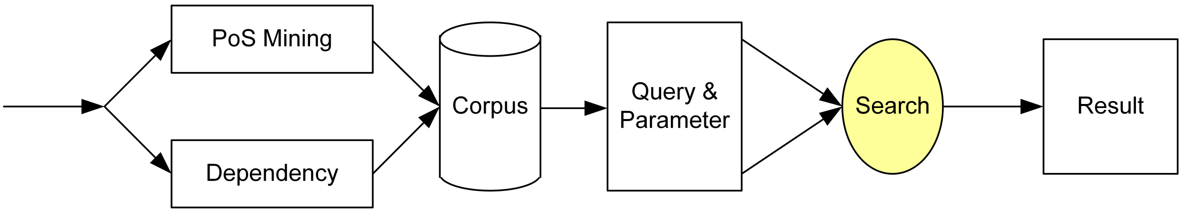
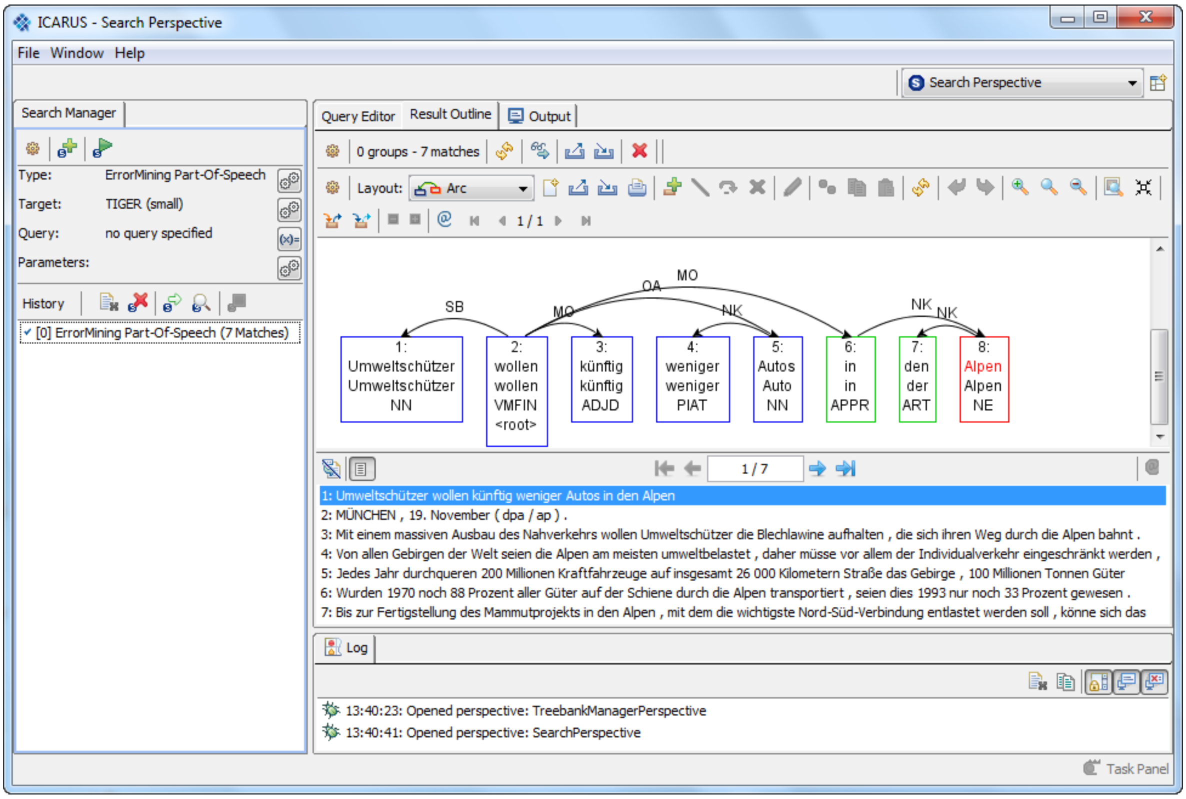


Figure 2.1: Workflow error mining

The illustration in Figure 2.2 show an instance of the error mining search. Before starting a new error mining search, in the upper left of the window, the type must be set (error mining dependency structure or part-of-speech). Afterwards the target corpus is specified.
In this example we did not specify a query. Finally the search parameter (detailed options are explained in Section 2.5.5). The program is very close to the error pipeline as described above to keep the error mining process as simple as possible.

The right part of the window shows the resulting dependency graph for the first sentence that include the variation 3-gram <in den Alpen, {APPR ART NE}>. The nucleus <Alpen> is colored red, <in den> is coloured green since it is part of the 3-gram but has no variation. Below the graph is a list containing all sentences with one n-gram. By clicking on the other sentences the corresponding result graph including the appropriate highlight will show up.

In the following Section 2.1 we give a general description of the error mining algorithm introduced by (Dickinson, 2005, p. 33-38). In Section 2.2 we talk about the use of n-grams and



Type: ErrorMining Part-Of-Speech
Target: TIGER (small)
Query: no query specified
Parameters:

✓ [0] ErrorMining Part-Of-Speech (7 Matches)

1: Umweltschützer wollen künftig weniger Autos in den Alpen
2: MÜNCHEN , 19. November (dpa / ap) .
3: Mit einem massiven Ausbau des Nahverkehrs wollen Umweltschützer die Blechlawine aufhalten , die sich ihren Weg durch die Alpen bahnt .
4: Von allen Gebirgen der Welt seien die Alpen am meisten umweltbelastet , daher müsse vor allem der Individualverkehr eingeschränkt werden ,
5: Jedes Jahr durchqueren 200 Millionen Kraftfahrzeuge auf insgesamt 26 000 Kilometern Straße das Gebirge , 100 Millionen Tonnen Güter
6: Wurden 1970 noch 88 Prozent aller Güter auf der Schiene durch die Alpen transportiert , seien dies 1993 nur noch 33 Prozent gewesen .
7: Bis zur Fertigstellung des Mammutprojekts in den Alpen , mit dem die wichtigste Nord-Süd-Verbindung entlastet werden soll , könne sich das

```
13:40:23: Opened perspective: TreebankManagerPerspective
13:40:41: Opened perspective: SearchPerspective
```

Figure 2.2: Screenshot Error Mining Search Perspective

their benefits for error mining. After that we will discuss our implementation for part-of-speech tags (Section 2.3) and dependency structure error mining (Section 2.4). In the final part of this chapter we will talk about the filters, query editor and further search parameters.

2.1 Error Mining Algorithm

Before we go into details we describe the general idea behind the algorithm described in Dickinson and Meurers (2003). The authors use a simple algorithm to compare two strings and find variation between pairs of <wordform, tag> (tag = e.g. part-of-speech). Variation occurs if one type has at least two different tags assigned <type,$\{tag_a, tag_b\}$> ($tag_a \neq tag_b$) in the same corpus. Wordforms with variation are named *variation nuclei* (uni-gram). The Example 2.1 shows a slice of a sentence and below each word the corresponding part-of-speech tag. The word *higher* is a variation nucleus since it has two different tags assigned. Our notation would be <higher, {JJR (adjective, comperative), RBR (adverb, comperative) }>.

(2.1)	*share*	*prices*	*closed*	***higher***	*in*
	NN	NNS	VBD	**JJR/RBR**	IN

Finding these variations is not limited to part-of-speech tags, the possibility for adapting the algorithm was shown for dependency structures by Boyd et al. (2008) or discontinuous structures proposed in Dickinson and Meurers (2005). The necessary changes to use the algorithm for dependency structures will be explained later in Section 2.4.

As we mentioned in chapter 1 using only uni-grams in the error mining process is not sufficient because of language ambiguity (shown in Example 1.4, 1.5 and 1.6). We need to find variation that occurs within the same context. If one word form occurs twice within the same context, and has at least two different tag assignments the probability that this was caused by erroneous tagging is higher than only looking at the nucleus itself. To discover varying tags in the same context Dickinson uses n-grams which we will explain in the next section.

2.2 N-Grams

For each sentence the algorithm tries to build the longest varying n-gram which is equivalent to the largest matching context. An n-gram is a sequence of tokens. The n-gram length grows for each iteration step. Depending on its size we speak of a uni-gram (1-gram), bi-gram (2-gram), tri-gram (3-gram) and so on. The (n+1)-grams are build incremental using the n-grams. The largest possible n-gram for each sentence is limited by the number of words per sentence. $SentenceLength = |Sentence|$

Dickinson (2005) states that if we can create the (n+1)-gram of a given n-gram and if this (n+1)-gram still full-fills the constraint that there is still variation within the (n+1)-gram, its sufficient to continue the algorithm using only the (n+1)-grams for the n+2 iteration.

Assume we have two different sentences A and B and $a_x = WordformAtIndex(x)$ as stated in 2.2:

$$\begin{aligned} &A : (a_1, ..., a_{|A|}) \\ &B : (b_1, ..., b_{|B|}) \end{aligned} \tag{2.2}$$

These two sentences contain one nucleus N (one type with at least two different tag assignments as shown in 2.3):

$$\begin{aligned} &N_A \in a_1, ..., a_{|A|} \\ &N_B \in b_1, ..., b_{|B|} \end{aligned} \tag{2.3}$$

Both nuclei have different tags but the same wordform.

$$Tag(a_{N_A}) \neq Tag(a_{N_B}) \text{ and } a_{N_A} = a_{N_B} \tag{2.4}$$

The first assumption in 2.4 ensures that the assigned tags are different for sentence A and B whereas the second ensures that the wordform is equal (definition for variation nucleus). The algorithm would try to find the largest common n-gram for sentence A and B. Therefore it extends both nuclei to the left (L) and/or right (R) until we reach the start/end of the sentence A or B (see 2.5) or until we have no longer the same context.

$$\begin{aligned}&\text{Sentence start: } L < N_A, L < N_B\\&\text{Sentence end: } R + N_A < |A|, R + N_B < |B|\end{aligned}\tag{2.5}$$

The largest varying n-gram (with the same context) we can find is:

$$n - gram_A = (a_{N_A-L}, ...a_{N_A}, ...a_{N_A+R}) = (b_{N_B-L}, ...b_{N_B}, ...b_{N_B+R}) = n - gram_B\tag{2.6}$$

It follows that every smaller *n-gram'* is part of the largest n-gram (see 2.7):

$$\begin{aligned}&\text{Sentence start': } L' < L < N_A, L' < L < N_B\\&\text{Sentence end': } R' < R + N_A < |A|, R' < R + N_B < |B|\\&n - gram'_A = (a_{N_A-L'}, ...a_{N_A}, ...a_{N_A+R'}) = (b_{N_B-L'}, ...b_{N_B}, ...b_{N_B+R'}) = n - gram'_B\\&\Rightarrow (n - gram'_A = n - gram'_B) \subseteq (n - gram_A = n - gram_B) \qquad \square\end{aligned}\tag{2.7}$$

We have shown that the resulting (n+1)-gram always covers the smaller n-gram (no information is lost). This is important when running the algorithm. It is sufficient when generating the (n+1)-grams to only use all n-grams from the immediately proceeding step, i.e. to create the (n+2)-grams using only the (n+1)-grams and so on.

N-grams are used to find words within the same context, and as we already mentioned before for larger contexts the probability that we have found an error is higher. This fact is useful when examining the results because it may be enough to start with the largest n-grams per sentence that the algorithm has found for one variation nucleus to detect errors. However, sometimes smaller n-grams must be reviewed as well. This may occur when no “number wildcards” (introduced in Section 2.5.2) are used and therefore sentences which contain the same words but different numbers no longer have the same context. We will show this during the evaluation in section 3.1.2.

The Example 2.8 shows that every (n+1)-gram contains its parent n-grams. We picked an 11-gram from the German TIGER Corpus (Example 2.8 a) with the variation nucleus (bold) “enger”. Below each word are the corresponding part-of-speech tags and the translation. The nucleus “enger” has two tag-assignments ADJD (adjective, adverbial or predicative) and ADJA (adjective, attributative). This 11-gram contains two variation 10-grams. They can be reconstructed by elimination of the first b) or last word c) of the variation 11-gram shown in Example 2.8 a).

(2.8) a)

insbesondere	*der*	*Gefahr*	*einer*	*weltweiten*	*Klimakatastrophe*	*,*	*besteht*
ADV	ART	NN	ART	ADJA	NN	$,	VVFIN
particularly	the	risk	of	global	climate catastrophe	,	there is

ein	***enger***	*Zusammenhang*
ART	**ADJD/ADJA**	NN
a	**close**	connection

b) der Gefahr einer weltweiten Klimakatastrophe , besteht ein **enger** Zusammenhang

c) insbesondere der Gefahr einer weltweiten Klimakatastrophe , besteht ein **enger**

2.3 Part-Of-Speech Error Mining

Part-Of-Speech tags are used for grammatical classification of words. The most simple tag assignments would be to classify words using the following classes: noun, verbs, adjectives or adverbs. However pos-tags used by linguists are wider partitioned and use sub-categories as well (e.g. verbs can be partitioned by their tenses, numerus,...). The decision which tag is assigned depends on the word definition (morphology) and the surrounding context (syntax).

Part-of-speech tags are assigned to every token (words and punctuation marks) within a sentence. Example pos-tag sets can be found in the appendix for both treebanks that we will use later on.

We start with the part-of-speech error-mining implementation (The dependency structures algorithm is slightly different. Details will be explained in Section 2.4). It is based on (Dickinson, 2005, p. 33-34) which is an instance of the a priori algorithm proposed in Agrawal and Srikant (1994).

During initialization (Alg. 2.1 line 2-6) the algorithm reads the whole corpus. All tokens are stored with their corresponding pos-tags. This is done by the *StoreTags()* method. The *StoreTags()* method distinguishes between three cases and works the following way:

1. For a given token there exists no entry within the *ngramlist.* We just add the <token,{tag}> pair to the *ngramlist* (line 15-18).
2. The <token,{tag}> combination is already in our list. Then we have to increase the occurrence count for the given *tag* (line 9-10).
3. The token is already in the *ngramslist* but the given *tag* was never assigned before (line 11-13). In this case the algorithm will expand the current tag-set with the *new tag* using the *ExpandTag()* method. Expanding the tag-set by a second *tag* leads to variation.

After the initialization and after each further n-gram processing step, the “variation filter” method (Alg. 2.2) is applied to the *ngramlist.* It checks if there is any word with more than one tag assignment (line 3). Other pairs (which have exactly one tag) are dropped because they have no variation. The resulting *ngramlist* contains all uni-grams (1-grams) of the given input corpus with at least two different tag assignments.

Algorithm 2.1 Initialization for part-of-speech Error Mining:
tagOccurenceCount (tOC), numberOfDifferentTags (nODT)

```
 1: $ngramlist = null$;

 2: procedure INITIALIZETAGS($corpus$,$parameters$)
 3:     for all $tokens \in corpus$ do
 4:         STORETAGS(token, postag)
 5:     end for
 6: end procedure

 7: procedure STORETAGS($ngramlist$,$token$,$postag$ )
 8:     if $token \in ngramlist$ then
 9:         if postag == AlreadyKnownTag then
10:             $tOC_{new} = tOC_{old} + 1$
11:         else
12:             $nODT_{new} = nODT_{old} + 1$
13:             EXPANDTAG($tag_{new}$)
14:         end if
15:     else
16:         $tOC = 1$;
17:         $nODT = 1$;
18:         ngramlist = ngramlist($token, tag, nODT, tOC$)
19:     end if
20: end procedure
```

The procedure "FilterLength" shown in Algorithm 2.2 is a crucial part of the error mining process and is called after every iteration. Basically this method checks if there is variation within the given n-gramlist. Variation is found if one word contains more than one tag ($numberOfDifferentTags(nODT) > 1$ line 2). For the (n+1)-gram generation only words with variation are interesting. All others are dropped (Alg. 2.2 line 5-6) from the result list for the next iteration.

(2.9) a) *interested*
VBN

b) *interested*
JJ

Looking at Example 2.9, the Algorithm 2.1 would store <interested, {VBN}> when the word "interested" occures for the first time and the algorithm continues with the following token. Lets assume we found a second token with the wordform "interested" in our corpus. First we check if there is already an n-gram for the specific token stored in the *ngramlist* (Alg. 2.1 line 8). Since we stored <interested, {VBN}> before, this is true. Now we verify if there is already the {JJ} tag assigned. Since we only added <interested, {VBN}> we cannot find the

Algorithm 2.2 Method checks if one n-gram has more than one tag assigned. Others were dropped. This is done using the numberOfDifferentTags (nODT) calculated before.

```
procedure FilterLength(ngramlist)
    for all ngram ∈ ngramlist do
        if nODT > 1 then
            Variation found ⇒ keep in ngramlist;
        else
            Only one tag assigned ⇒ no variation ⇒ remove n-gram from ngramlist;
        end if
    end for
end procedure
```

{JJ} tag. The algorithm enlarges the tag-set to <interested, {VBN, JJ}> (line 13) and also increase the number of varying tags we have found for the token “interested”. Assuming there is no further varying tags for “interested” the algorithm would keep <interested, {VBN, JJ}> and would pass the *ngramlist* to the *FilterLength()* method (Alg. 2.2) as soon as we reached the last token in the corpus.

To generate (n+1)-grams the algorithm uses the constructed *ngramlist* as input. Then the *GenerateNGrams()* (Alg. 2.3) checks if the n-grams can be extended to the left/right (line 4-9). This is done at most once for each iteration as long as we have not reached the first/last token in the sentence. The resulting new (n+1)-grams are stored in a new *ngramlist*. Then the *FilterLength()* method (Alg. 2.2) is executed since we only want to keep n-grams that still have variation. If the filtered *(n+1)-gramlist* contains at least one (n+1)-gram the *GenerateNGrams()* method (Alg. 2.3) is executed again iteratively using the filtered *(n+1)-gramlist* as a new input for the (n+2)-grams.

Algorithm 2.3 Method uses a former *ngramlist* as input and tries to extend the existing n-grams to the left/right until we reach the sentence boundaries

```
procedure generateNGrams( ngramlist )
    (n+1)-ngramlist = null;

    for all $ngram \in ngramlist$ do
        if $!(token - 1 = reachedSentenceBegin)$ then
            $ngram = token - 1 + token$
            StoreTags( (n+1)-ngramlist, ngram, tag)
        else if $!(token + 1 = reachedSentenceEnd)$ then
            $ngram = token + token + 1$
            StoreTags((n+1)-ngramlist, ngram, tag)
        else
            Can't expand the current ngram ⇒ continue with next ngram
        end if

        FilterLength( (n+1)-ngramlist )

        if (n+1)-ngramlist > 1 then
            generationNGrams( (n+1)-ngramlist )
        else
            Finished and Show Result
            ShowResult(
        end if)
    end for
end procedure
```

2.4 Dependency Structure Error Mining

Dependency grammars (DG) have become more popular over the last years since they are especially useful for analysing languages with free word order (e.g. German, Czech). Modern dependency grammars are based on the seminal work of Tesnière (1959). A theory how to work with DGs was introduced by Mel'çuk and A. (1988). The hierarchical structure of DGs is represented as trees that only consist of terminal nodes (words). The structure is determined by the relation between two words, one word (head) governs the other word (dependent).

An example for a dependency tree is shown in Figure 2.3 a): both *Mary* and *John* depend on *kissed.* The fact that no word governs *kissed* makes it head of the sentence. The relations (edges) are labeled with syntactic functions. They may vary depending on the treebank / language. For example in Figure 2.3 we annotated the same sentence twice using the "Penn Treebank II Tagset (Santorini, 1990)" for the English "Mary kissed John." (2.3 a) and the "STTS Tagset (Schiller et al., 1999)" for the translated German sentence "Mary küsste John." (2.3 b). The English sentence got the following labels assigned "Marry(SBJ=Subject) kissed(root=head) John(OBJ=Object) .(P=Punctuation)".
The German sentence is labeled "Marry(SB=Subject) küsste(root=head) John(OA=Accusative Object) .(–)". As we can see both sentences have the same edges (direction, source/target node) but since the tagset differs the edge labels are not the same.

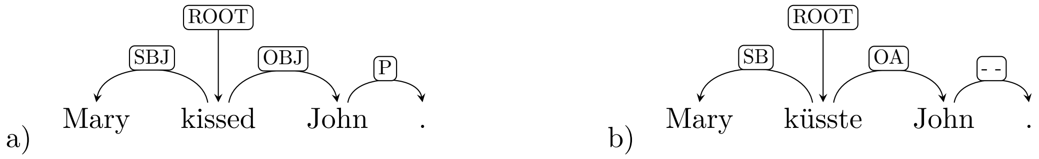


Figure 2.3: Simple dependency structure example (English/German) that shows the different dependency labels across different treebanks.

In this thesis we implemented the error detection mechanism for dependency treebanks introduced in Boyd et al. (2008) but there are other approaches for (automated) error detection (and correction) (e.g. Volokh and Neumann (2011) approach which is similar to van Halteren (2000)). A GUI-based solution was introduced by Agarwal et al. (2012b) to detect errors and correct them on Hindi Dependency Treebanks based on the work of Ambati et al. (2010a), Ambati et al. (2011) and Agarwal et al. (2012a).

We start discussing the differences between part-of-speech and dependency error-mining. In general the algorithm works in the same way but the data structure changes. For part-of-speech we only had a simple <token, {tag}> mapping. A dependency edge has a source node and a target node as shown in Figure 2.4. Therefore we change the structure to <token1 token2, {tag}>. Additionally to encode the edge direction a suffix is added to the tag (_Left / _Right) to determine if we found an incoming or outgoing edge. This is implemented as described in (Boyd et al., 2008, p. 14).

The Algorithm 2.4 shows the pseudo code for dependency structure error-mining. In comparison with the pos-initialisation Algorithm 2.1 the dependency structure initialization is slightly

Algorithm 2.4 Initialization for dependency structure error mining:

```
 1: $ngramlist = null$;

 2: procedure initializeTags($corpus$,$parameters$)
 3:     for all $words \in corpus$ do
 4:         dependencyItem = getDependencyTag(token);
 5:         relationTag = dependencyItem.getTag();
 6:         token = dependencyItem.getToken();
 7:         StoreTags(ngramlist, token, relationTag)
 8:     end for
 9: end procedure

10: procedure getDependencyTag($token$)
11:     if $token = rootnode$ then
12:         //nothing to do because a rootnode has no incomming edges
13:         return $(token, “root'')$;
14:     else
15:         if $token = targetnode$ then
16:             getSourceNode(
17:             $)relationTag =$ getRelationTag$() + \_Left$
18:             $token = token + sourcenode$
19:             return $(token, relationTag)$;
20:         else
21:             getTargetNode(
22:             $)relationTag =$ getRelationTag$() + \_Right$
23:             $token = token + targetnode$
24:             return $(token, relationTag)$;
25:         end if
26:     end if
27: end procedure
```

different. The *InitializeTags()* method change because we have to determine the specific relation tag (including the edge direction). Therefore before calling *StoreTags()* we create the appropriate tag. This is done using the *GetDependencyTag* method (Alg. 2.4 line 4). First we check if the input node is a root node. Since we use as a mapping scheme <node1 node2, {tag}>, a root node will result in <rootnode “empty”, {tag}>. If the input node is not a root node, we further check if there are any incoming or outgoing edges (line 12). We then up retrieve the specific edge label (line 13 / line 18) and add the suffix to encode the edge direction. The computed *token* / *relationTag* are passed to the *StoreTags()* method which works the same as described above in the Algorithm 2.1 for part-of-speech error mining.

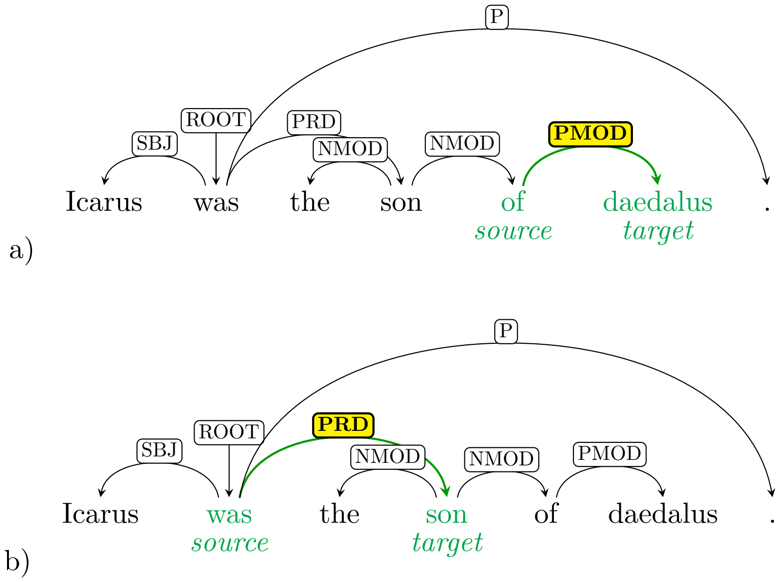


Figure 2.4: The figure shows two dependency trees for the sentence: *Icarus was the son of daedalus*. Source/Target nodes are coloured green. Example b) shows a gap between the source- and target node

Dependency nodes are not necessarily immediate neighbours. There may be a gap between source- /target-node as shown in Figure 2.4 b). Running the *InitializationTags()* method would result in the following tags: <of daedalus, {PMOD_Right}> and <was son, {PRD_Right}>. As we already mentioned <was son, {PRD_Right}> shows a gap with the size of one caused by the word “the”. While executing the *GenerateNGrams()* method shown in Algorithm 2.3 we first try to close those gaps. The algorithm tries to include the words between two nodes before it starts expanding the n-gram using outlying words. This is done until it closed the gap or until it is not possible to close the gap any further because of varying words within. Then it will continue the normal way and build new n-grams using the outlying words. For Example 2.4 the algorithm would first produce the n-gram <was the son, {PRD_Right}> (then the gap is closed) before the n-grams <Icarus was son, {PRD_Right}> or <was son of, {PRD_Right}>

2.5 Filters, Query Editor and Search Options

In this section we talk about the implemented filters. The result of the error mining algorithm may contain n-grams which look like errors but they are not. The error mining plug-in provides heuristics to filter out some of these “false” hits. These heuristics are implemented as introduced by (Dickinson, 2005, p. 34-39). The remaining hits, after filtering the results using a heuristic, have a higher probability to be errors. As a side effect they could save time when searching errors. Afterwards we introduce the query editor and give a description on its features and how it can be used when error mining a treebank. Finally we take a look at the implemented search options that could be set before starting an error mining process.

2.5.1 Long Context Filter

We talked about n-grams in Section 2.2 and the fact that a (n+1)-gram covers its parent n-grams. The author (Dickinson, 2005, p. 39-43) pointed out that for the WSJ n-grams sized $n \geq 6$ deliver a sufficient context to determine the correct tag. This means when looking at 6-grams the provided context is wide enough for a human annotator to decide whether there is an error or not. Our plug-in provides this functionality when viewing the results. The user can set a minimum/maximum n-gram size. Then the algorithm will exactly complete n-passes, and show all n-grams it has found within the nth iterations and that fulfil the minimum constraint.

2.5.2 Number Wildcard Replacement

When the *number wildcard replacement* filter is enabled the algorithm checks for every word-form during the error mining process if the current word is a number. This is done using a regular expression that flags all words where the first letter is a number (0...9). These words will be replaced with a "[NumberWildcard]" (NWC) later. As we will show later in Chapter 3 this filter is useful: It provides the error mining algorithm with the capability to compare strings that contain different numbers and treat them equally in order to find variation within the non-number word-forms. For example by using this filter we were able to compare the strings *<since 1985>* and *<since 1990>* because both are mapped to *<since [NWC]>*. The replacement is only conducted within the internal error mining data-structure. The filter does not change any treebank data.

2.5.3 Fringe Heuristic

The fringe heuristic is used to filter n-grams where the nucleus occurs at the start/end of the n-gram. This is done because when the nucleus is surrounded by words the probability that we find an error is higher. For example see Figure 2.5. The figure shows three iteration steps (N=1, N=2 and N=3) of the error mining algorithm for the variation nucleus <Put, NN, VBN> (dashed with a red border), every token that is coloured orange is part of the current n-gram. Starting with N=1 the algorithm first detects the variation for the word "put". For the N=2 step we can only extend the nucleus to the right as it is at the sentence start and therefore no extension to the left is possible. The resulting bi-gram is <Put option, NN, VBN NN>. In iteration n=3, we can only extend the bi-gram to the right. The resulting 3-gram is <Put option in, NN, VBN NN IN>. We may continue this way until the sentence end. But this would end up in a n-gram where only the first token differs assuming that "put" is the only variation nucleus within the whole sentence.

However we want to find variation for one nucleus that occurs multiple times within the same context. When the nucleus is at the start/end of a sentence there is no surrounding context to the left/right. With the heuristic enabled the 3-gram <Put option in, NN, VBN NN IN> would be deleted from the result list and therefore not passed to the n=4 iteration step because of its missing left context. This ensures that we have the nucleus embedded by at least one word left and right when the fringe heuristic is switched on.

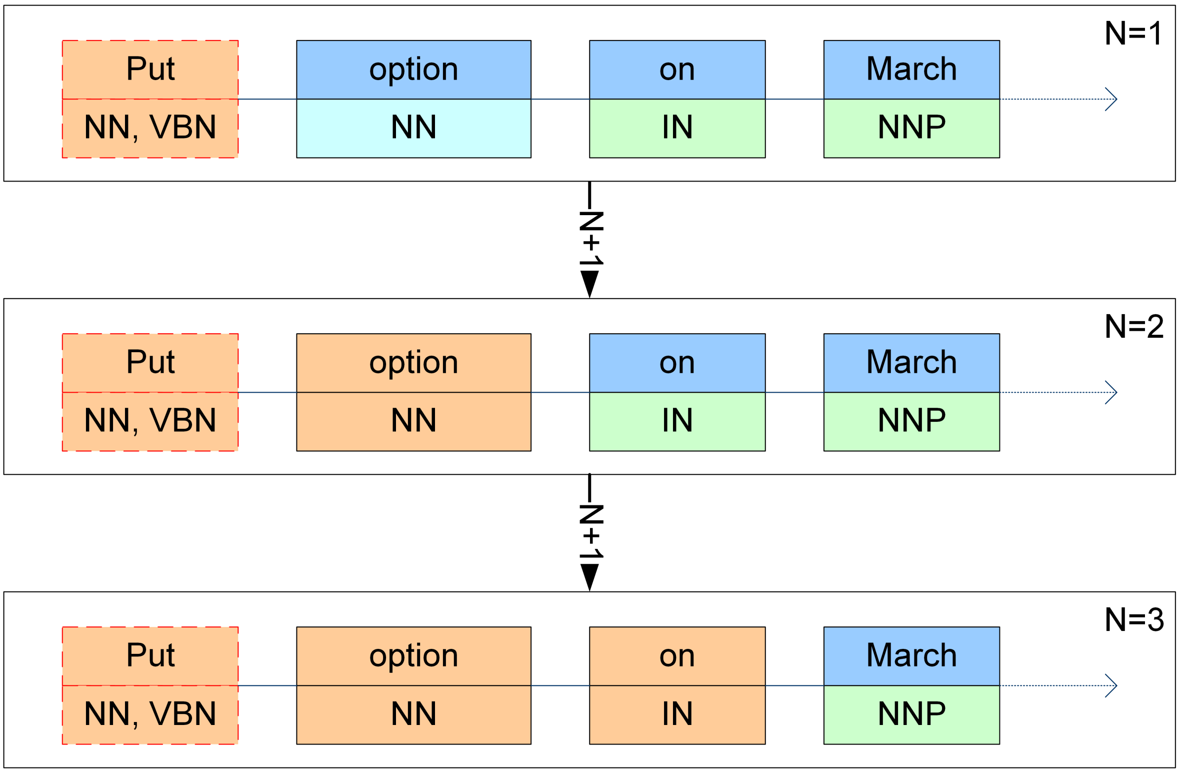


Figure 2.5: Fringe Heuristic Example. Variation nucleus is dashed and everything that is part of the current n-gram orange

To explain our fringe heuristic implementation we use Figure 2.6 as a running example where we show three steps of a error mining algorithm (N=1, N=2 , N=3). Every token that is part of the current n-gram is coloured orange.

The initialization is completed and we found one nucleus <Children, {NN, NNP}> (nucleus border red dashed). This nucleus is used as input for the next iteration (N=2). The algorithm extends the uni-gram to the left and right. As a result we get two bi-grams <National Children, {NNP} {NN, NNP}> and <Children 's, {NN, NNP} {POS} >. At this point we cannot say whether one of these two bi-grams should be filtered out because we only have two iteration (left/right) done so far and therefore the nucleus must be at the fringe. Using the two bi-grams as an input for the next step would create the two 3-grams shown in Example 2.10:

(2.10)

a)	*National*	*Children*	*'s*
	{NNP}	{NN, NNP}	{POS}

b)	*Children*	*'s*	*Cancer*
	{NN, NNP}	{POS}	{NNP}

Note that with the expansion "<National Children>" we have already reached the first word in the sentence and therefore there is no further extension to the left possible.

When the fringe heuristic enabled it will remove the 3-gram <Children 's Cancer, {NN, NNP} {POS} {Cancer}> (Figure 2.6 n=3 purple dashed) from the result list before the next iteration step because the nucleus is at the fringe of the given 3-gram. Assuming there are no further 3-grams the only input for n=4 would be <National Children 's, {NNP} {NN, NNP} {POS}>. Nevertheless we do not lose hits, this operation just prunes the search space because during the n=4 step the Algorithm 2.3 would produce the 4-gram <National Children 's Cancer,{NNP} {NN, NNP} {POS} {NNP}> anyway.

Our implementation uses this observation and lets the user decide at which point he wants to enable the fringe heuristic. For all algorithm iterations $n \geq 3$ the fringe heuristic can be triggered with the result that our search space is reduced for the next iterations since only the necessary n-grams will be kept where at least one nucleus is not at the fringe.
Assume we have a 5-gram containing five words with their corresponding tags as shown in Example 2.11:

(2.11)

w_1	w_2	w_3	w_4	w_5
t_{1a}	t_{2a}	t_{3a}	t_{4a}	t_{5a}
t_{1a}	t_{2b}	t_{3a}	t_{4a}	t_{5b}

The fringe heuristic implementation will only compare the tag-sequence $t_{2a}t_{3a}t_{4a}$ vs $t_{2b}t_{3a}t_{4a}$. If we find variation within these tags we keep the result. We do not have to compare the whole tag-sequence since even if there is variation as in the example above $t_{5a} \neq t_{5b}$ these variation would be at the fringe anyway and therefore deleted. It is sufficient only to compare the $t_2...t_4$ tags for the given 5-gram. Using a 3-gram we would only compare t_2, for a given 10-gram only $t_2...t_9$ and so on.

2.5.4 Query Editor

The query editor provides the functionality to group tags together, rename tags or exclude tags from the error mining search. Figure 2.7 shows the editor view. It is organized in three parts. On the left side there are buttons to create/edit or delete a single query. In the middle there is an overview over all specified queries represented as a list. Below are three buttons to manage the queries supporting save/load or reset.
The capability of saving a query to an extensible mark-up file (xml) and load it again later is useful if the user specifies a query and wants to use it later in different corpora. Using reset will delete all specified query items.

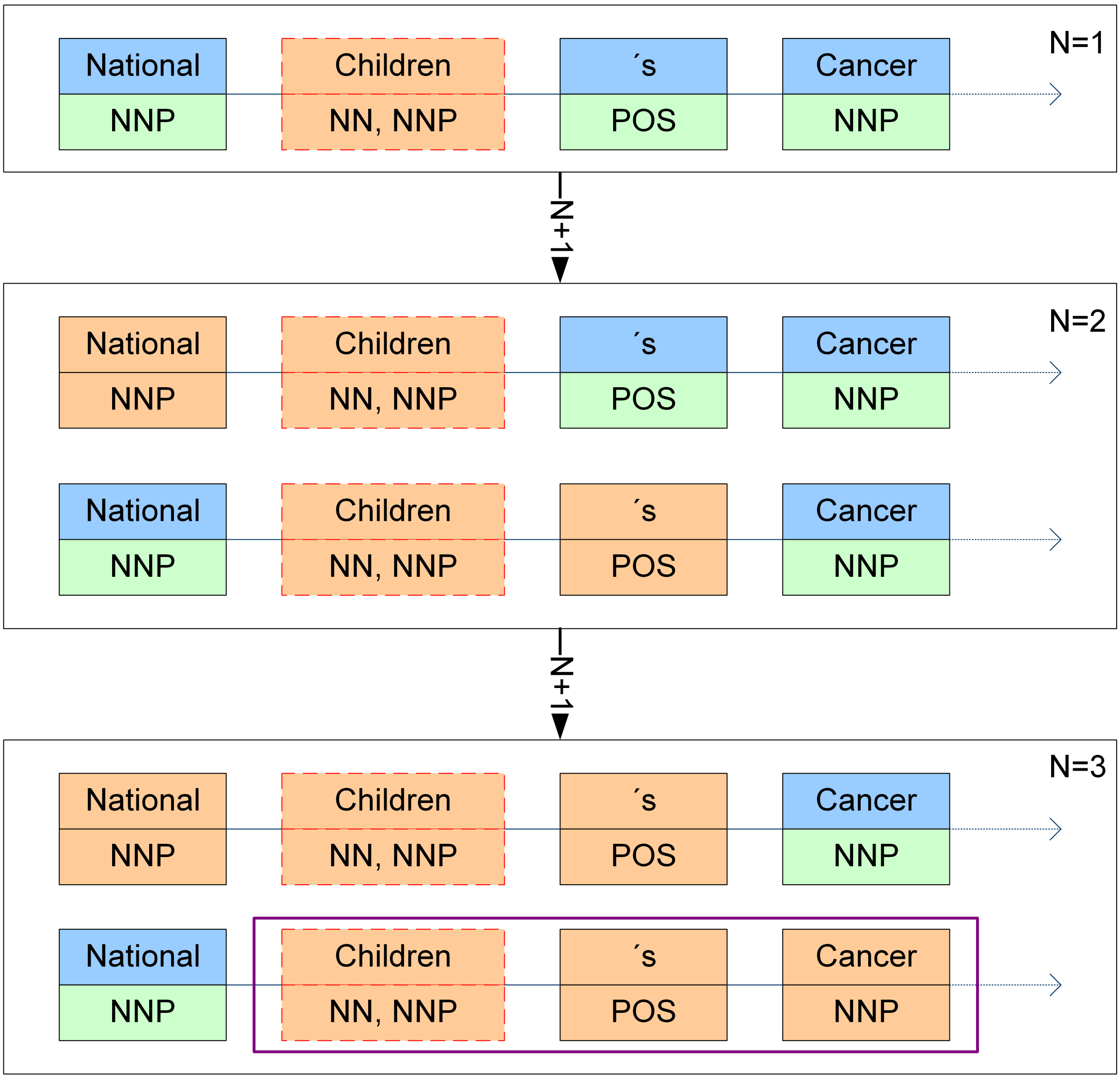
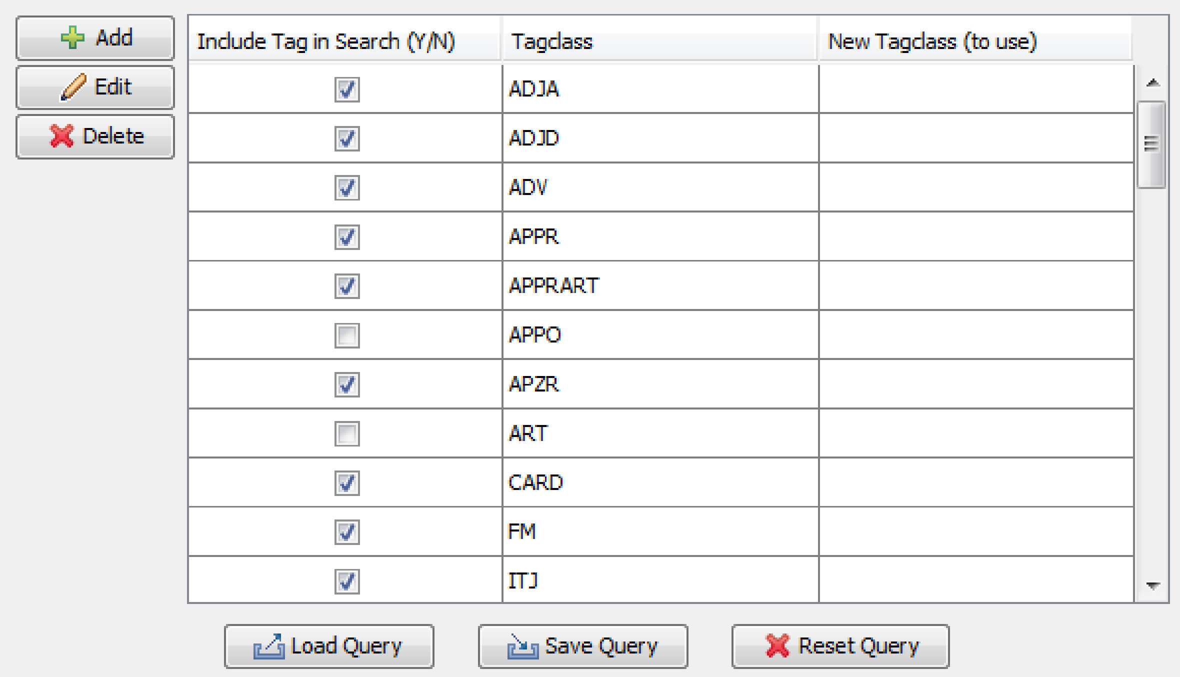


Figure 2.6: Fringe Heuristic Example. Variation nucleus is dashed, n-grams orange. The 3-gram that would be filtered when *fringe enabled* has a purple border and occur in step N=3

A single query item always has the following format: <Include Tag in Search, Tagclass, new Tagclass>. It is processed in the following way as described in the query editor algorithm (Alg. 2.5). For a given <token, {currentTag}> combination the algorithm checks if one matching query exists (line 4). A matching query is a query where $Tagclass = currentTag$. If this check is successful the "include check" is next. Using the "include tag option" (boolean) the user



Include Tag in Search (Y/N)	Tagclass	New Tagclass (to use)
☑	ADJA	
☑	ADJD	
☑	ADV	
☑	APPR	
☑	APPRART	
□	APPO	
☑	APZR	
□	ART	
☑	CARD	
☑	FM	
☑	ITJ	

Figure 2.7: Query Editor Screenshot

can specify if one tag shall be considered or ignored. All tags that are ignored are mapped onto a [TagIgnoredTagclass]. This parameter has priority over the new tag assignment. But if the current tag is not found within the query list it is neither ignored nor does it get a new tag assigned and the algorithm just continues the normal way taking the current tag. The benefit of this design is that there is no need to put the whole tag-set into the query system. Otherwise if there is only one tag missing it would never be recognized, and therefore never show up in the variation. The field *Tagclass* and *new Tagclass* use strings as a type. The *Tagclass* itself has to be unique, e.g it is not possible to use <true, *NN*, NN1> and <true, *NN*, NNE>. A invalid *Tagclass* will be signalled by a error message.

Assume that there is a query match (Alg. 2.5 line 2), and furthermore the tag shall be included (line 5). Then the last query step takes place. The query will return either the current tag, when there is no new tag assigned (line 6-7) or the new tag (line 8-9).

Example 2.12 a) with "VBN" as the currentTag (the tag we look up in the query list) is used to illustrate the mechanism we described above. Using the query shown in Example 2.12 a) will use the "VB" tag for all occurrences of "VBN". No nucleus will contain any "VBN" tag since it was replaced by "VB".
If the user does not want to change the tag query Example 2.12 b) will do so. The algorithm would find the "NN" tag within the query item list but since there is no new tag assigned

Algorithm 2.5 Query Editor Algorithm

```
queryItemList;
The queryItemList includes all specified user querys

procedure QUERY(currentTag)

    if $currentTag \in queryItemList$ then
        if $include == true$ then
            if $newTag == null$ then
                return currentTag
            else
                return newTag
            end if
        else
            tag shall not be include drop <token,tag> and continue with the next
        end if
    else
        continue normal way
    end if
end procedure
```

(null) it will return the “NN” tag.
The last Example 2.12 c) will drop all occurrences of “NE” and replace them using the *[TagIgnoredTagclass]* . The resulting n-grams will not contain any “NE” tag. The defined “new tag” NN will of course show up since the include operator only affects the “old tag”.

(2.12) a) <true, VBN, VB>
b) <true, NN, null>
c) <false, NE, NN>

The xml structure of a query file is shown in Listing 2.1. Each *Queryitem* represents one constraint. The include flag is works as explained above. The “Tagclass” must be unique so editing should be done through the query editor, which is able to perform the appropriate checks, and not manually. “Value” represents the new tagclass if there was a tagclass defined before.

Listing 2.1 Example XML Query File showing all three format types: Include (boolean), Tagclass (= current Tag) (string), Value (= new Tagclass) (string)

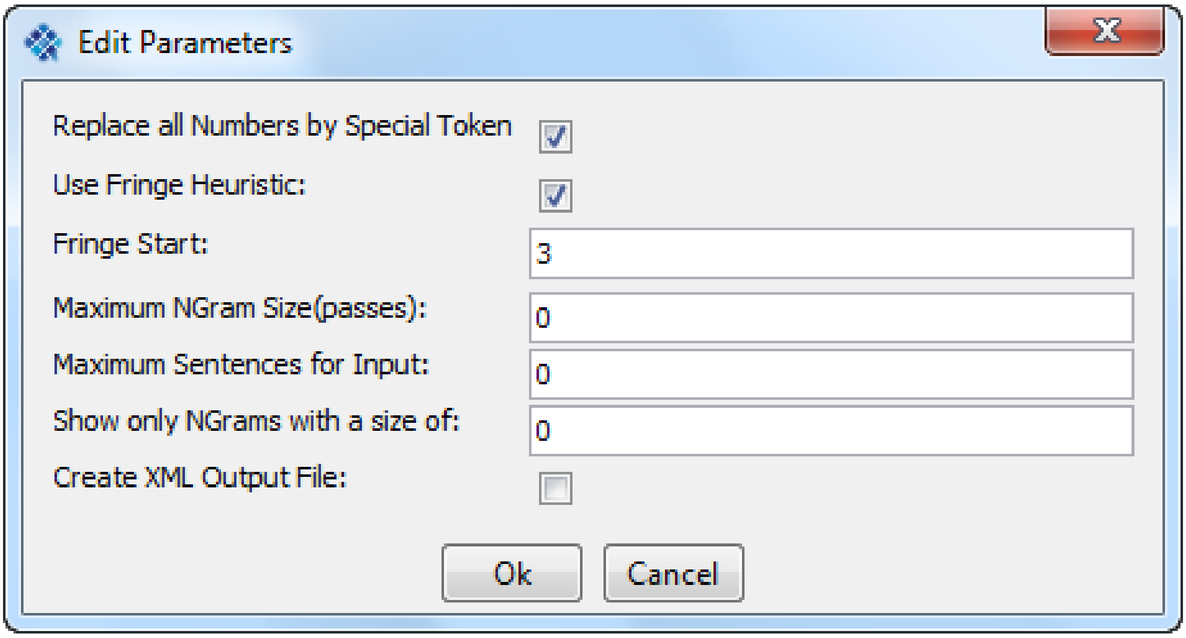
```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<NGram-Query>
<Queryitem Include="true" Tagclass="VBN" Value="VB"/>
<Queryitem Include="true" Tagclass="NN" Value=""/>
<Queryitem Include="true" Tagclass="NE" Value="NN"/>
<Queryitem Include="true" Tagclass="ADJA" Value="ADJ/ADJA"/>
</NGram-Query>
```

The query editor can be used to create a mapping between tags without changing the current corpus data. We believe this kind of exploring a corpus may help finding ambiguity classes for part-of-speech as proposed in Dickinson (2007). It may also be useful for dependency function labels. If the user is only interested in a special phenomenon a query can be used to filter n-grams, for example the distribution of <NN vs NE vs XY> for the German TIGER Corpus. Another possible use is to map tags to one class (e.g. all verbs in a "verb-class") and then run the error mining. It may be easier to find mistakes for words that should be tagged as a verb but apparently are not. Last but not least it could be used to compare tagging across different treebanks. Assume treebank A got 60 tags and treebank B only 48. Using the mapping we can try to map the 60 tags to our 48 tags and compare the results afterwards. The only restriction is that this only works in one direction, limiting the treebank tag set that has more tags. Introducing a richer tagging scheme does not work.

2.5.5 Error Mining Options

Using parameters allows the user to take influence on the results. The currently available parameters are shown in Figure 2.8.

- **Replace all Numbers by Special Token:** This option enables the *number replacement filter* as described in Section 2.5.2.
- **Use Fringe Heuristic:** the fringe heuristic we described earlier in Section 2.5.3 will not change the longest n-grams, as long as their nucleus is not at the fringe. Nevertheless using the fringe heuristic is recommended. It gives better results as Dickinson showed in (Dickinson, 2005, p. 40-43) because it guarantees that we have surrounding context. Eliminating fringe items can first be done in the third n-gram generation pass. Before that we just have uni-grams (first-pass) and bigrams (second-pass). For both cases holds that the nuclei must be at the fringe. But the concrete starting point for *enabling the heuristic* is open to the user.
- **Maximum NGram Size (passes):** Another filtering option is the limitation of the n-gram size (size = algorithm iterations). By default this parameter is zero which is equivalent to ∞ . Therefore by default the algorithm collects all n-grams it is able to find. If the value was changed to e.g. 5 the algorithm will execute the n-gram generation (Alg. 2.3) at most five times. Hence the maximum size for the n-grams that can be found within the result are 5-grams.



Replace all Numbers by Special Token ☑

Use Fringe Heuristic: ☑

Fringe Start: 3

Maximum NGram Size(passes): 0

Maximum Sentences for Input: 0

Show only NGrams with a size of: 0

Create XML Output File: □

Figure 2.8: Search Parameters for NGrams

- **Maximum Sentences for Input:** The sentence limitation is used to limit the number of sentences that are used for the error mining. Starting at sentence one until the specified value x. For example with a limit of 10,000 at most the first 10,000 sentences of the specified corpus will be used during the error mining process. This option may be useful if there is a corpus with a lot of errors and the user first wants to correct a few sentences or if the system would run out of memory when computing on all sentences. But using this option has a strong influence on the results and should be used carefully, because limiting the input data may leak the variation for one word. By default this value is "0" (zero) and the engine will use all sentences of the given corpus.

- **Show Only NGrams with a (minimum) size of:** Even when the fringe heuristic is enabled the results will still contain uni-/bi-grams. Using the "Show only NGrams with a size of" option allows the user to filter the resulting n-grams. For example if the value is set to "1", the resulting list will contain 2-, 3-, n-grams, ...

- **Create XML Output File:** Using the "Output to File" option creates an xml-formatted file with the format show in Listing 2.2. It contains information about the word-forms, tags, tag-count and highlight information. It is formatted in a human-readable way so that its possible to do error detection even without the graphical support of the error mining plug-in.

Listing 2.2 Example XML Result File

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<nGrams>
  <WordForm form="manufacturing" nGram="1">
    <PoSTag count="1" tag="NN">
      <Sentence begin="8" end="8" nucleiCount="1" nucleiStartIndex="8" sentenceNr="7">
        <NucleiIndex>8</NucleiIndex>
      </Sentence>
    </PoSTag>
    <PoSTag count="1" tag="VBG">
      <Sentence begin="3" end="3" nucleiCount="1" nucleiStartIndex="3" sentenceNr="14">
        <NucleiIndex>3</NucleiIndex>
      </Sentence>
    </PoSTag>
  </WordForm>
</nGrams>
```

2.6 Source code

The whole project was written in Java 1.7. For the concrete Java implementation please check the binaries[1] which are freely available under the GNU General Public License, version 3.

[1] http://www.ims.uni-stuttgart.de/data/icarus.html

3 Evaluation

In Chapter 2 we described the general error mining approach, followed by the description of our own implementation and the new features we added to the process. To check how the algorithm performs we made an evaluation on German and English.

The evaluation was done running four different error mining configurations. *Fringe heuristic* will be executed if switched on and then always in the fourth (4-gram) iteration. The second parameter we used is the *number wildcard replacement.* If *number replacement* option is switched on the algorithm determines if a word begins with a number (0...9). All numbers are replaced using a special number wild card (NWC). Using the *number replacement* we are able to compare constructions like <since 1985> and <since 1990> because they both will be mapped to <since [NWC]>. Varying these options we get the following four different error mining configurations.

Fringe heuristic *on*, number replacement *on* (F; NWC)

Fringe heuristic *off*, number replacement *on* (no F; NWC)

Fringe heuristic *on*, number replacement *off* (F; no NWC)

Fringe heuristic *off*, number replacement *off* (no F; no NWC)

In the section for part-of-speech error mining (Section 3.1) we present our results for the German TIGER and Penn Treebank. Additionally for the TIGER Corpus we performed a manual result evaluation. For the English Penn Treebank an extensive manual evaluation was done by (Dickinson, 2005, p. 39-44). The Dependency structure results are discussed in Section 3.2. In the final Section 3.3 we compare the results we have found during the pos / dependency error mining process.

3.1 Part-Of-Speech Tag Error Mining

3.1.1 PoS Error Mining Results for the German TIGER Corpus

The TIGER Corpus contains 50,472 sentences consisting of 888,239 tokens which have 87,423 different word-forms. In the following part we will discuss the results of our pos error mining process. A complete table consisting of all result data can be found in Appendix A.7. During the evaluation we will take a closer look especially on results with *enabled fringe heuristic* (described in Section 2.5.3) and *enabled number replacements.*

The largest n-gram we have found using the error mining options *fringe heuristic on* and *enabled number replacement on* (no query was specified) contains 56 tokens and is shown in Example 3.1. It has one variation nucleus *<ebenso, {KON, ADV}>*. The KON (coordinate conjunction) was found in sentence No. 14948 and ADV (adverb) in sentence No. 18113. In total the sentence contains 32 different uni-grams.

(3.1) Diese Ökonomie des Vermeidens läßt sich prinzipiell auch auf die Wärme- und Primärenergiemärkte übertragen, zum Beispiel auf die energetische Sanierung des Gebäudebestandes, wo das Marktvolumen auf rund [NWC] Milliarden Mark geschätzt wird; **ebenso** auf die Vermeidung ökologisch bedenklicher Stoff- und Materialflüsse und damit auf die Begrenzung von Abfall, Ressourcen und Mobilität.

In Example 3.2 we have two variation nuclei <bis> and <zu>. They were tagged in one sentence both ADV (adverb) and in the other sentence APPR (preposition, circumposition left). The [NWC] occurred during the number replacement process.

(3.2)	6-gram:	*sollen*	*künftig*	*Zinserträge*	***bis***	***zu***	*[NWC]*
		VMFIN	ADJA	NN	**APPR**	**APPR**	CARD
		VMFIN	ADJA	NN	**ADV**	**ADV**	CARD

(english: will in the future interest income up to [NWC])

Sentence (occurs twice): Wie berichtet, *sollen künftig Zinserträge* ***bis zu*** *6000* Mark bei Ledigen und 12 000 Mark bei Verheirateten steuerfrei bleiben.

In Example 3.3 we show a 6-gram with the variation nucleus <wie> tagged KOKOM (comperative conjunction) and PWAV (adverbial interrogative or relative pronoun). The PWAV assignment for this example is wrong.

(3.3)	6-gram:	*doppelt*	*so*	*hoch*	***wie***	*in*	*den*
		ADJD	ADV	ADJD	**KOKOM**	APPR	ART
		ADJD	ADV	ADJD	**PWAV**	APPR	ART

(english: twice as high **as** in the)

{KOKOM} Sie liegt laut Unicef *doppelt so hoch* ***wie*** *in den* USA, wo zehn Morde auf 100 000 Todesfälle kommen.

{PWAV} So beziffert das Institut für Arbeitsmarkt- und Berufsforschung die Gesamtzahl der Erwerbslosen heute auf ungefähr sechs bis sieben Millionen - *doppelt so hoch* ***wie*** *in den* Statistiken der Arbeitsverwaltung ausgewiesen.

To evaluate the German TIGER Corpus we used two different parameters fringe heuristic (F) and Number Wild Card (NWC) in every combination resulting in four different configurations shown in Table 3.1. Looking at row 1 we see that using the number wildcard (F; NWC and no F; NWC) in column two and three we find 4059 uni-grams instead of 4148 (column four and five) where we did not use the number replacement option. This is exactly what we expected since replacing numbers results in a smaller set of word types. This phenomenon can be observed within the next rows as well. For 8-grams or larger the difference between *NWC*

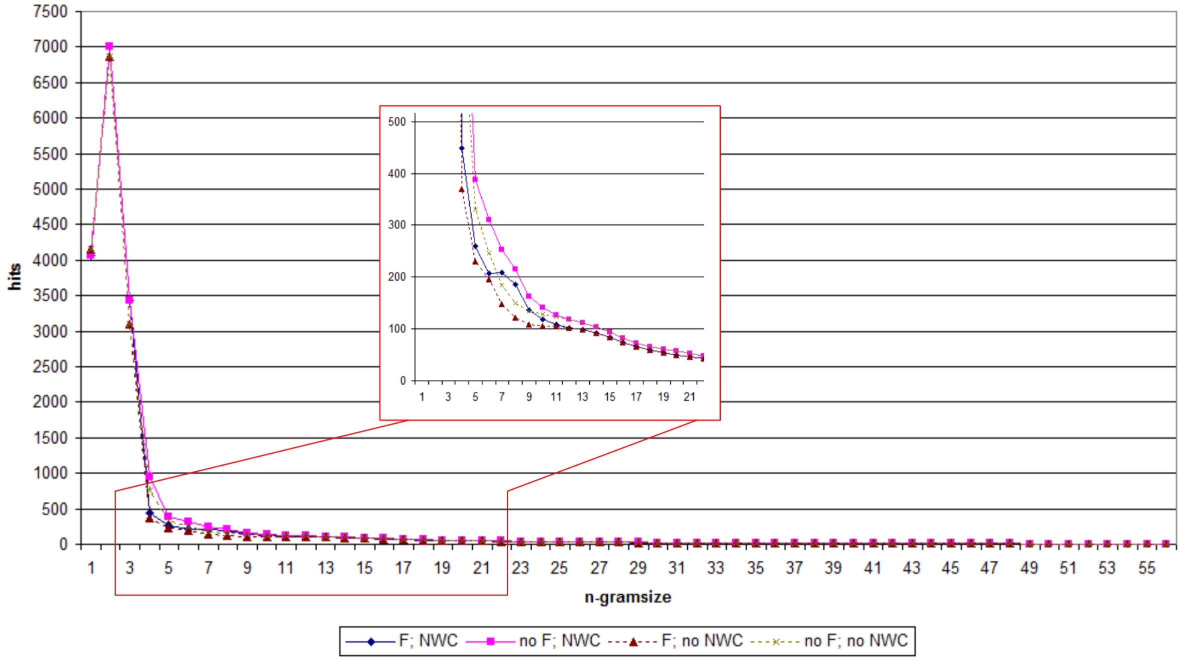


Figure 3.1: Result for the German TIGER Corpus visualizing all four error mining configurations we used during the evaluation (fringe heuristic on, number wildcards on), (fringe heuristic off, number wildcards on), (fringe heuristic on, number wildcards off) and (fringe heuristic off, number wildcards off) (x-axis = n-gramsize and y-axis = number n-grams found)

enabled and *NWC disabled* slightly decreases. When reaching the 36-gram all of the 4 error mining configurations contain the same resulting n-grams for the following iterations. This implies that also the size of the longest n-gram we found does not vary no matter which error mining parameters were set.

When the fringe heuristic is turned on we always execute it after the third iteration. Comparing column 2 (F; NWC) and 3 (no F; NWC) we find out that for n=4 we only have 448 4-grams left when the *fringe heuristic* is enabled as opposed to 948 grams with disabled *fringe heuristic*. The same can be observed for column four (F; no NWC) and five (no F; no NWC) with the corresponding values 369 vs 758. This can be monitored for the following n-grams (n>4). If the *fringe heuristic* is disabled we always have more n-grams within the result set. Nevertheless when reaching the 36-gram border the result n-gram count will not differ any more. This is also a phenomenon we expect since both parameters *fringe heuristic* / *number wild card* decrease the result n-grams. The fringe heuristic removes unwanted results and the NWC reduces the token count from the start because all numbers are mapped together. Anyway we were able to find the longest n-gram with any of the four configuration.

We illustrate this in Figure 3.1 where for all n>36 (x-axis) all four configurations continue to have an equal number of results (y-axis). We zoomed in on the part with the greatest differences between *fringe heuristic* and *non-fringe.* As we can see the fringe heuristic curves are always below the non-fringe curves until 36 when all coverage. This observations must not hold for other treebanks. When using the *number replacement* option, there may be cases, where the largest n-gram that can be found for *enabled number wild-cards* is larger compared to the largest n-gram with *disabled number wild-cards.*

N-Gram	F; NWC	no F; NWC	F; no NWC	no F; no NWC
1	4059	4059	4148	4148
2	7004	7004	6870	6870
3	3434	3434	3093	3093
4	448	948	369	758
5	260	388	230	331
6	207	310	195	246
...	...	...	...	...
36	20	21	20	21
37	20	20	20	20
38	19	19	19	19
...	...	...	...	...
54	3	3	3	3
55	2	2	2	2
56	1	1	1	1
57	-	-	-	-

Table 3.1: Result for the German TIGER Corpus showing parts of the four error mining configurations we used during the evaluation (fringe heuristic on, number wildcards on), (fringe heuristic off, number wildcards on), (fringe heuristic on, number wildcards off) and (fringe heuristic on, number wildcards off)

Querying Close Tag Classes (German TIGER Corpus)

Using only problematic tag classes in an n-gram query may help finding errors faster because the resulting dataset is much smaller and easier to verify. We built a query where only variation within NN (common noun), NE (proper noun) and XY (non-word containing non-letter) is observed and every other tag is replaced by a [NULL-TAG]. This is useful if the user only wants to take a look at a specific linguistic phenomenon or if he is only interested in the distribution of specified query tags.

Running the algorithm (*fringe heuristic enabled* and *number wildcard enabled*) we get the following results (illustrated in Table 3.2) where we show a slice of the resulting n-grams (The complete result list containing all n-grams can be found in Appendix A.8). Using only (NN + NE) as a query we received 1335 1-grams, 1018 2-grams, 474 3-gram and 169 4-grams. Including XY to the query these values change to 1347 1-grams, 1023 2-grams, 476 3-gram and 169 4-grams. As we can see in Table 3.2 the hits for the <NN + NE + XY> query in column three are slightly higher compared to the n-grams in column two. This occurs because we are recognizing a third tag (XY) in column three, which result in more variation. Reaching n=4 both queries have the same result-set.

N-Gram	Hits (NN + NE)	Hits (NN + NE + XY)
1	1335	1347
2	1018	1023
3	474	476
4	169	169
5	159	159
6	140	140
...	...	...
29	2	2
30	1	1
31	-	-

Table 3.2: Result for TIGER Corpus using two different queries: NN, NE and NN, NE, XY executed with the error mining configuration (fringe heuristic on, number wildcards on)

The largest n-gram we found (*fringe heuristic on*, *enabled number replacement on* and using the query <NN, NE, XY>) is shown in Example 3.4. It contain 30 tokens with the variation nucleus *ECR-Tornados* tagged NN (common noun) within sentence No. 33703 and NE (proper name) in sentence No. 34192 case. The correct tag is NE.

(3.4) Das schließt den Einsatz von Kampfflugzeugen (**ECR-Tornados**) aus ", heißt es dazu in einer ergänzenden Formulierung, die der neue Parteichef Oskar Lafontaine durchgesetzt hat.

A 3-gram we found during the process was <in den Alpen>. It was tagged four times {APPR ART NN} (Example 3.5 a) whereas the tagging {APPR ART NE} (Example 3.5 b) only occurs two times.

(3.5) a) {APPR ART NN}

- Zudem haben in den zurückliegenden Wintern viele Wintersportler mit der Schneesicherheit *in den **Alpen*** schlechte Erfahrungen gemacht.
- Bis zur Fertigstellung des Mammutprojekts *in den **Alpen***, mit dem die wichtigste Nord-Süd-Verbindung entlastet werden soll, könne sich das Verkehrsverhalten der Menschen geändert haben.
- Die zwei Spuren im Schnee finden, wenn man will, Anschluß an Fernloipen oder enden nach ein paar Kilometern vor Hütten, in denen der Glühwein auf dem Herd dampft und die Gäste wie auf den Almstadeln *in den **Alpen*** singen: Bergvagabunden sind wir.
- Das hieß aber auch zunächst: Zoff mit den Alpinskifahrern und Teilen der Davoser Bevölkerung, Politikern inklusive, die den "Chaoten" - wie überall *in den **Alpen*** - weder Platz im Lift noch Raum auf den Pisten einräumen wollten.

b) {APPR ART NE}

- Nach Angaben des DNR findet ein Viertel des gesamten Welttourismus *in den **Alpen*** statt.
- Umweltschützer wollen künftig weniger Autos *in den **Alpen***

Another example is the 6-gram containing two news agency abbreviations "Agence France-Presse (afp)" and "Deutsche Presse-Agentur (dpa)". The 6-gram contains the following tokens *(afp / dpa)*. We found variation within *afp* and *dpa* 13 times. They were both tagged as NE and in 2 times as XY ("non-word"). The results are even worse when we look at the corresponding uni-grams: the results in Table 3.3 show that *dpa* is tagged 559 as NE and 60 times XY. Same for *afp* where we have 223 NE and 45 XY instead of the correct pos tag NE.

Token	NE (Count)	XY (Count)
dpa	559	60
afp	223	45

Table 3.3: Tag distribution between part-of-speech tag NN and XY for the uni-grams <dpa> and <afp> using the error mining configuration (*fringe heuristic on*, *number wildcards on*) with the query <NN, NE and XY>. (Neither dpa or afp was tagged NN therefore NN is missing in this table)

The examples above showed that even when it should be clear to distinguish between NN, NE and XY there are many errors within the corpus. Running the error mining process using these three tags deliver a result set targeting these classes. In turn the resulting data could be easily reviewed manually.

Manual PoS Error Mining Result Evaluation (German TIGER Corpus)

In this section we manually check if the algorithm finds errors and how much of them are false positives. We picked the first 115 of the 207 6-grams we have found in total during the error mining process (*fringe heuristic on*, *number replacement on*). Then we manually check for all nuclei within the 115 n-grams if the variation is caused by erroneous tagging. If the tag is erroneous we try to determine the correct tag. In total we count 119 variations. This is caused by the effect that as long as we have find an n-gram that includes two or more nuclei we count each nucleus separately if they are separated by at most one word that has no variation. Table 3.4 shows all variations we find sorted by their number of occurrence.

Count	variation a	variation b
41	NN	**NE**
22	**NE**	XY
7	APPR	ADV
6	ADV	**ADJD**
4	**VVPP**	ADJD
3	**ADJA**	PIS
3	**APPR APPR**	ADV ADV
3	**ART**	CARD
3	**PDS**	ART
3	PDS	**PDAT**
3	PIS	PIAT
3	**PWAV**	PROAV
3	FM	**NE**
2	**KON**	ADV
2	PIS	**ADV**
2	NE NE	**FM FM**
1	ADV	**PTKVZ**
1	ADJA	**NE**
1	ADJD	**ADJA**
1	FM FM	**NE NE**
1	FM FM FM	**NE NE NE**
1	**NN**	XY
1	**PRELS**	**ART**
1	**VAPP**	VAINF

Table 3.4: Results German TIGER Corpus manual evaluation for 119 sampled 6-grams (fringe heuristic on, number wildcards on) correct tag is bold if we could determine it

The high numbers for {NN vs NE} and {NE vs XY} are not that surprising although the decision between NN and NE is complicated. As we found out during the tag query in Section 3.1.1 there is a lot of variation especially in tagging of the news agency abbreviations (dpa, afp,...), which is reflected in these numbers. In total we are able to detect errors and determine the correct tag for 113 of the given 119 nuclei, a precision of 94.94%.
For NN vs NE variation is shown in (Example 3.6). We found the exact sentence twice but with the variation <Mittelmeer, {NN , NE}> the correct pos-tag is NE (proper noun).

(3.6) 6-gram: *des* ***Mittelmeeres*** *spezialisiert, zeigt aber*

{**NE**, NN} Der jetzt eröffnete Unterwasser-Zoo ist auf die Unterwasserwelt *des* ***Mittelmeeres*** *spezialisiert, zeigt aber* auch tropische Arten.

Example 3.7 shows inconsistency within the verb <worden>, the sentence occurs twice in the corpus.

(3.7) 6-gram: *eines EU-Mitgliedsstaates zugelassen* ***worden*** *sein.*

{**VAPP**, VAINF} Das Saatgut muß von der zuständigen Behörde *eines EU-Mitgliedsstaates zugelassen* ***worden*** *sein.*

We found 5 nuclei (4.20%) where the tagging is inconsistent but we are not able to find the correct tag even using the tagging rules. In Example 3.8 and 3.9 we show five different sentences where we cannot determine, using the 6-gram, the correct tag (6-gram is bold, nucleus red). Nevertheless there should be only one correct solution how the word <plus> must be tagged but the tagging guide lines do not cover this. Note that we would not be able to find these n-grams without the *number replacement*.

(3.8) 6-gram: *[NWC] Milliarden (* ***plus*** *[NWC] Prozent*

a) {APPR} Hoechst strich, auch dank Sondererträgen, *3,4 Milliarden (* ***plus*** *103 Prozent*) ein, Bayer "nur" 3,3 Milliarden (39 Prozent mehr).

b) {ADV} Bayer schaffte *3,3 Milliarden (* ***plus*** *39 Prozent*), und BASF bleibt nach einer rasanten Aufholjagd nur 200 Millionen dahinter zurück.

(3.9) 6-gram: *Mark (* ***plus*** *[NWC] Prozent)*

a) {APPR} 337 000 *Mark (* ***plus*** *18 Prozent)* setzte SNI zuletzt pro Beschäftigten um.

b) {APPR} Die Krankensparte erwartet (mit Pflege) 31,9 Milliarden *Mark (* ***plus*** *12,8 Prozent).*

c) {ADV} Von Januar bis September verdiente Hoechst vor Abgaben an den Fiskus 3,4 Milliarden *Mark (* ***plus*** *103 Prozent).*

Finally the nucleus <die, {PRELS, ART}> (0.84%) was no error at all but genuine ambiguity. The two sentences are shown in Example 3.10 (6-gram is bold, nucleus red). Looking at the first sentence (Example 3.10 a) the PRELS (substituting relative pronoun) tag is correct. For the other sentence b) the ART (definite or indefinite article) part-of-speech tag is correct. This example shows clearly that we cannot rely on the fact that the algorithm only predict errors.

Such false errors may be removed if the fringe heuristic is changed, e.g. in the current implementation the fringe heuristic ensure that we have exactly one word on each side of a nucleus. If we change the heuristic in a-way that there must be at least two words on each side this variation would no longer be in the result set because on the left we have: <Asylbewerber ,> in sentence a) and <Tokio ,> in sentence b) and therefore no longer the same context.

(3.10) 6-gram: *, **die** in der Nacht zum*

a) {PRELS} Drei afrikanische Asylbewerber*, **die** in der Nacht zum* Sonntag im ostsächsischen Zittau nach einer Messerstecherei festgenommen worden waren, haben offenbar in Notwehr gehandelt.

b) {ART} Ministerpräsident Ryutaro Hashimoto sagte in Tokio*, **die** in der Nacht zum* Samstag erreichte Einigung bringe lediglich eine Atempause.

These results are not completely comparable to (Dickinson, 2005, p. 39-44) because he evaluated on the Wall Street Journal Data and we used the German TIGER Corpus. Additionally he sampled 125 n-grams within ($6 < n < 224$) of his 7141 distinct non-fringe n-grams and we took 115 non-fringe 6-grams into evaluation. The precision he detected for the WSJ was 92.8%.

3.1.2 PoS Error Mining Results for the Penn Treebank Data

To evaluate our implementation on English we used the Penn Treebank training data-set (section 2-21), which is part of the Penn Treebank. The Penn Treebank training data consist of 39,279 sentences that contains 958,166 tokens with 35,050 different word-forms.

Looking at the results in table 3.6 we can observe the impact of the fringe heuristic. The fringe heuristic almost halved the 4-grams when *switched on.* We got 3845 hits (column 2 row 4 (no F, NWC)) versus 1959 hits within (column 1 row 4 (F, NWC)). The same observation can be made when the number wildcard is turned off and fringe heuristic on/off. We found (no F; no NWC) 3124 and (F; no NWC) 1609 n-grams.

Another interesting fact is that when using the *number replacement* option we are able to find n-grams up to 64 while when switching off this option we only find n-grams with a maximum length of 55. The Figure 3.2 and 3.3 visualize the results (x-axis = n-gramsize and y-axis = number grams found). As we can see the curves are further apart from each other than before in the TIGER results. However the fringe heuristic curves are still below the non-fringe curves as in the TIGER Corpus. As mentioned before in Figure 3.3 there are no further n-grams after 55.

(3.11) LONDON **LATE EURODOLLARS** : [NWC] [NWC] % to [NWC] [NWC] % one month ; [NWC] [NWC] % to [NWC] [NWC] % two months ; [NWC] [NWC] % to [NWC] [NWC] % three months ; [NWC] [NWC] % to [NWC] [NWC] % four months ; [NWC] [NWC] % to [NWC] [NWC] % five months ; [NWC] [NWC] % to [NWC] [NWC] % six months

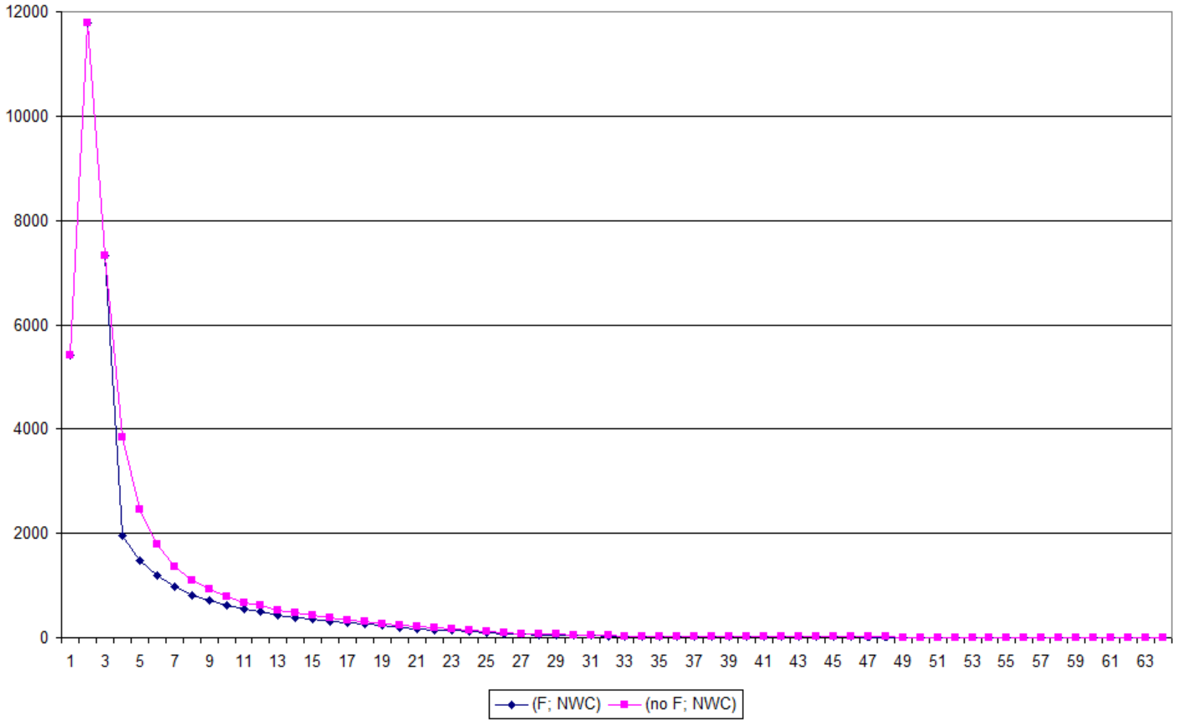


Figure 3.2: Result for the Penn Treebank training data showing two curves for the configuration (fringe heuristic on, number wildcards on) and (fringe heuristic off, number wildcards on) (x-axis = n-gramsize and y-axis = n-number grams found)

Taking a closer look at the longest n-grams. In Example 3.11 we can see the longest n-gram we have found in the Penn Treebank training data with *fringe heuristic on*, *number replacement on*. No query was specified.

Every [NWC] stands for a number replacement which was done during error mining process. We detect two variation nuclei <LATE EURODOLLARS> with four different taggings shown in Table 3.5. Six times we discovered {JJ NNS}, two times {RB NNS}, {JJ NNPS} and {RB NNPS} both occurred only once. In total the sentence contain 50 variation nuclei (uni-grams).

	{JJ NNS}	{RB NNS}	{JJ NNPS}	{RB NNPS}
<LATE EURODOLLARS>	6	2	1	1

Table 3.5: Table showing the longest n-gram with its varying tags that we have found during the error mining process for the Penn Treebank training data using the options *fringe heuristic on*, *number replacement on*. No query was specified.

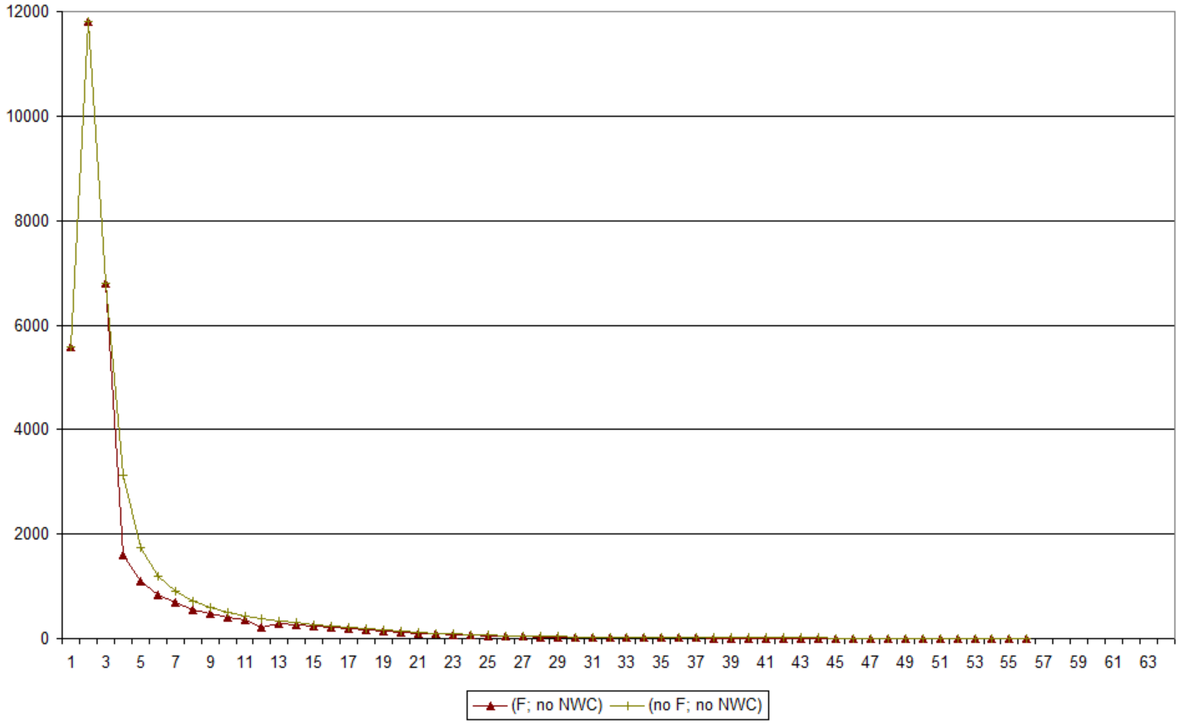


Figure 3.3: Result for the Penn Treebank training data showing two curves for the configurations (fringe heuristic on, number wildcards off) and (fringe heuristic on, number wildcards off) (x-axis = n-gramsize and y-axis = number n-grams found)

The size of the largest n-gram changes when we *disable number replacement*. As we can see in Example 3.12 (*fringe heuristic on*, *number replacement off* and no query specified). There are still two variation nuclei but only two different tags for <LATE EURODOLLARS> remain: {JJ NNS} and {RB NNPS}. Nevertheless the other variation still occurs within the result set but their variation n-grams no longer cover the same context because there are different numbers within the sentences. We would miss these variation when only looking at the largest n-grams. For the same reason we were also no longer able to find all hits ({JJ NNS} occurs six times when using number wild cards) without we only retrieve one hit count for the 55-gram. We believe that this shows very clearly the benefit using number wild cards.

(3.12) LONDON **LATE EURODOLLARS** : 8 11\/16 % to 8 9\/16 % one month ; 8 11\/16 % to 8 9\/16 % two months ; 8 11\/16 % to 8 9\/16 % three months ; 8 5\/8 % to 8 1\/2 % four months ; 8 9\/16 % to 8 7\/16 % five months ; 8

The resulting Table 3.6 also shows that it takes much longer until the number of n-grams converges compared to the TIGER results. Using *number replacements* one step before the final 55-gram the results collapse, that means that not until this point the algorithm would pass the same 54-grams to the next iteration step. For column 2 (F; NWC) and column 3 (no F; NWC) we find the the same observation. The passed n-gram results are not the same until the 63-gram which is again one step before the final 64-gram iteration.

N-Gram	F; NWC	no F; NWC	F; no NWC	no F; no NWC
1	5423	5423	5578	5578
2	11783	11783	11819	11819
3	7318	7318	6795	6795
4	1959	3845	1609	3124
5	1480	2456	1105	1744
6	1201	1795	840	1200
...	...	...	...	...
54	8	9	2	2
55	7	8	1	1
56	6	7	-	-
57	5	6		
...	...	...		
62	2	3		
63	2	2		
64	1	1		
65	-	-		

Table 3.6: Result for the Penn Treebank training data showing parts of the four error mining configurations we used during the evaluation (fringe heuristic on, number wildcards on), (fringe heuristic off, number wildcards on), (fringe heuristic on, number wildcards off) and (fringe heuristic on, number wildcards off)

Querying Close Tag Classes (Penn Treebank)

For the English corpus we built a query which includes all of the following verb-classes: VB (verb, base form), VBZ (verb, 3rd person singular present), VBP (verb, non-3rd person singular present), VBD (verb, past tense), VBN (verb, past participle) and VBG (verb, gerung or present participle). We will use V* as an abbreviation. Secondly we will switch on the parameters *fringe heuristic* and *number replacement*.

The 34-gram showed in Example 3.13 is the longest we found. The variation nucleus (bold) is <following, {VBN, JJ}>. We did not include {JJ} in our search but we are still able to find this variation. The reason is that all other tag-classes are mapped to the same [TagIgnoredTagclass] (e.g. DT, EX, NN,... and especially JJ). Running the algorithm we are able to detect this variation since *following* was tagged {VBN} and therefore we compare it with the [TagIgnoredTagclass] as well. When looking up the sentence in our result table we find the {JJ} variation (because the [TagIgnoredTagclass] does not affect the corpus data). In total we counted two times <following, {VBN}> versus six occurrences of <following, {JJ}>. This shows how the query mechanism could be used to fing errors within tagclasses.

(3.13) The **following** were among yesterday's offerings and pricings in the U.S. and non - U.S. capital markets, with terms and syndicate manager, as compiled by Dow Jones Capital Markets Report:

Another case where we found variation within the V*-class is shown in the 5-gram Example 3.14. Here we found the variation nucleus <contribute, {VB, VBP}>. Both tagging occurs exactly one. We determine that the correct tag for contribute is {VB}.

(3.14) 5-gram:	*administration*	*contends*	***contribute***	*to*	*the*
	[TagIgnored]	VBZ	**VB**	[TagIgnored]	[TagIgnored]
	[TagIgnored]	VBZ	**VBP**	[TagIgnored]	[TagIgnored]

{VB} Officials familiar with the meeting said Mr. Bush cited the policy as an example of the sort of congressional requirements the *administration contends* ***contribute*** *to the* failure of such covert actions as this month's futile effort to oust Panamanian dictator Manuel Noriega.

{VBP} The existence of the policy became known after Bush disclosed it to seven GOP senators last week, citing the plan as an example of congressional requirements the *administration contends* ***contribute*** *to the* failure of covert actions, officials said.

Looking at Example 3.15, we have the nucleus <stand, {VBP, VBN}>. The tag distribution was as follows: we counted 8 hits for <stand, {VBP}> in contrast to 4 hits by <stand, {VBN}>. However {VBP} is the correct tag.

(3.15) 5-gram:	*that*	***stand***	*as*	*milestones*	*of*
	[TagIgnored]	**VBP**	[TagIgnored]	[TagIgnored]	[TagIgnored]
	[TagIgnored]	**VBN**	[TagIgnored]	[TagIgnored]	[TagIgnored]

Sentence occurs 12 times with different taggings for <stand>: (During its centennial year, The Wall Street Journal will report events of the past century *that* ***stand*** *as milestones of* American business history.)

A more general query might be useful where all V* tags are also internally mapped to a new V*-class. Then we would no longer detect variation within (VBP, VBN, VB...). We would only find variation between the V*-class and all other possible tags, which where mapped to the [TagIgnoredTagclass].

3.2 Dependency Structures Error Mining

In the following section we evaluate the dependency structure algorithm (Section 2.4) using the German TIGER and English Penn Treebank. The evaluation was done using the *number wildcard replacement* and switch it on/off. Varying these options we get the following two different error mining configurations.

number replacement *on* (NWC)

number replacement *off* (no NWC)

3.2.1 Dependency Structure Error Mining Results for the German TIGER Corpus

The error mining settings are: *disabled fringe heuristic* and *enabled number replacement*. Running this configuration the largest n-gram we found contains 55 tokens and is shown in Example 3.16 (*fringe heuristic off*, *number replacement on* no query specified). We found two variations both shown in Figure 3.4. The first variation is between the first *auf* linked to *ebenso conjunct (CJ)* or *auf coordinating conjunction (CD)*. The ingoing edge lead to another variation. For sentence a) the first <auf> is head of <ebenso> and <ebenso> is dominating the second <auf> (CJ). In the second sentence b) the first <auf> is head of the second <auf> which is head of the word <ebenso> with the edge label modifier (MO) .

(3.16) Diese Ökonomie des Vermeidens läßt sich prinzipiell auch auf die Wärme- und Primärenergiemärkte übertragen, zum Beispiel **auf** die energetische Sanierung des Gebäudebestandes, wo das Marktvolumen auf rund [number-wildcard] Milliarden Mark geschätzt wird; **ebenso auf** die Vermeidung ökologisch bedenklicher Stoff- und Materialflüsse und damit auf die Begrenzung von Abfall, Ressourcen und Mobilität.

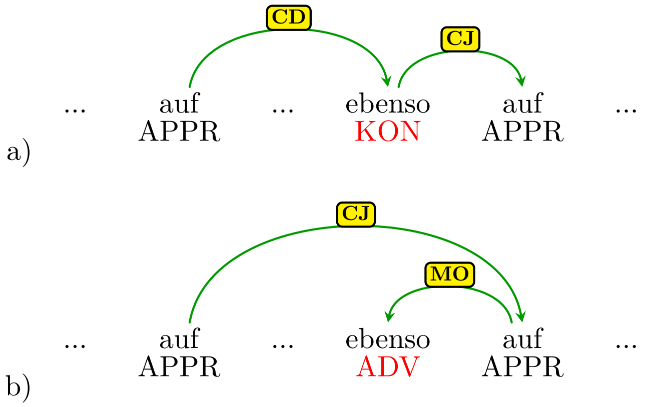

CD
CJ
... auf APPR ... ebenso KON auf APPR ...
a)
CJ
MO
... auf APPR ... ebenso ADV auf APPR ...
b)

Figure 3.4: Variation within the depenency structure and additional pos variation for <ebenso> (coloured red)

But this is not the only variation. We also have pos-tag variation (coloured red). And as we showed in Section 3.1 for part-of-speech error mining we found the same sentence with the nucleus *ebenso* tagged KON (coordinate conjunction) and ADV (adverb) in the other case. This shows clearly how errors within a treebank may depend on each other. Since the dependency were generated using the given morphology and part-of-speech tag information these dependency variation might be the result of erroneous pos tagging.

However using dependency structure error mining we also find n-grams that we would not be able to find running only the pos error mining algorithm. For example the 6-gram shown in Example 3.17 and 3.5. For every different tag we picked one sentence (for APL_L we only show 1 out of 6).

We found three differently tagged dependency structure labels all of them shown in Figure 3.5 relevant source/target nodes are coloured green. The nucleus <Gemeinschaft GUS, {APP_L}> was found 6 times in contrast to <Gemeinschaft GUS, {OP_L}> and <Gemeinschaft GUS, {PAR_L}> that both occur only once. The dependency structure labels may vary although the pos-tags are the same in Example 3.17.

(3.17) 6-gram: ***Gemeinschaft*** *Unabhängiger* *Staaten* *(* ***GUS*** *)*
Commonwealth of Independent States (**CIS**)

{APP_L} Russische Staatsbürger in anderen Republiken der ***Gemeinschaft*** *Unabhängiger Staaten (* ***GUS*** *)* müßten zwar beschützt werden, doch dürfe dies nur mit politischen Mitteln geschehen, sagte Jelzin in Moskau.

{OP_L} Freilich: Bei allein 28 Frauenorganisationen in St. Petersburg sind die rund 200 in der ***Gemeinschaft*** *Unabhängiger Staaten (* ***GUS*** *)* geförderten Tacis-Projekte nur ein Tropfen auf die schwache Pflanze Bürgergesellschaft.

{PAR_L} Und die Allianz - sicher ist sicher - widerstand nicht der Möglichkeit, ihre Grenzen bis an die der ***Gemeinschaft*** *Unabhängiger Staaten (* ***GUS*** *)* auszudehnen.

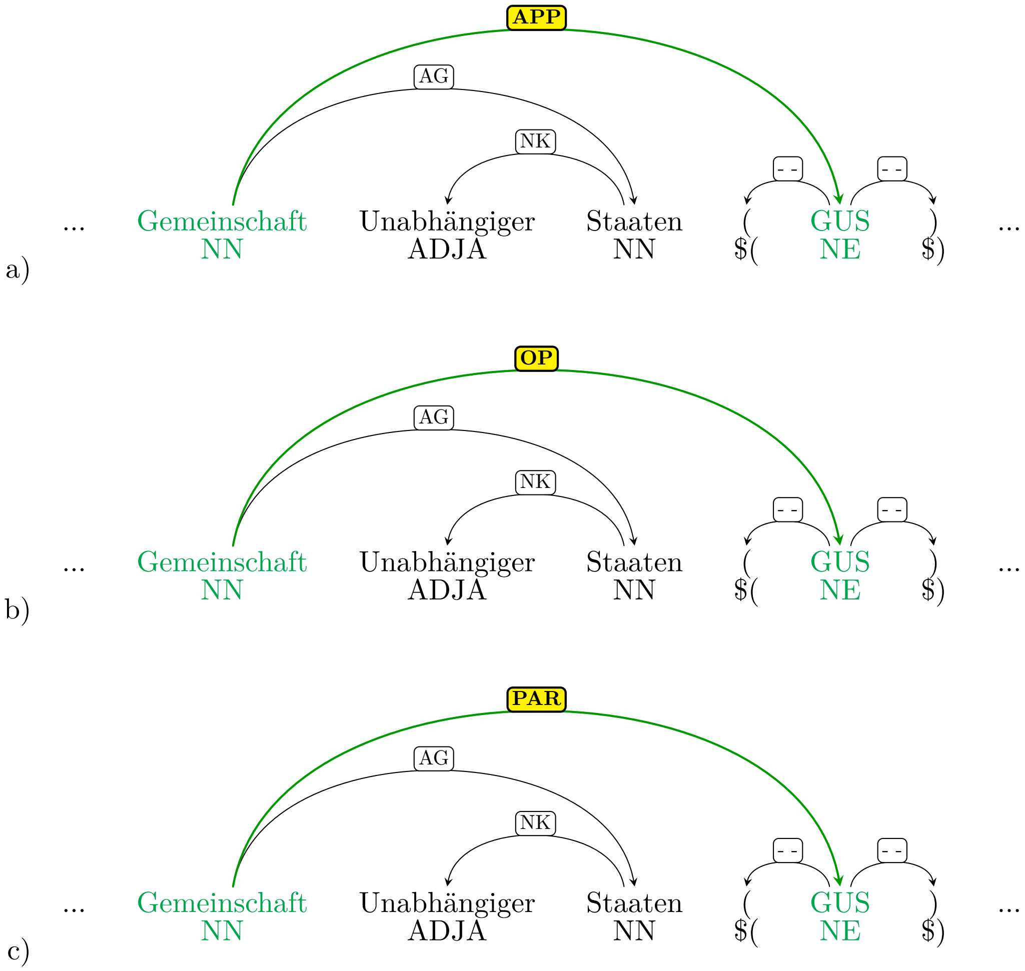

APP
AG
NK
- -
- -
... Gemeinschaft Unabhängiger Staaten (GUS) ...
NN ADJA NN $(NE $)
a)
OP
AG
NK
- -
- -
... Gemeinschaft Unabhängiger Staaten (GUS) ...
NN ADJA NN $(NE $)
b)
PAR
AG
NK
- -
- -
... Gemeinschaft Unabhängiger Staaten (GUS) ...
NN ADJA NN $(NE $)
c)

Figure 3.5: The figure shows the dependency structure of the German sentence slice *Gemeinschaft Unabhängiger Staaten (GUS)* The relevant nodes where variation occurs are coloured green. Below each word-form is the corresponding part-of-speech tag. Part-of-speech tag variation is coloured red.

3.2.2 Dependency Structure Error Mining Results for the Penn Treebank

The 47-gram in Example 3.18 is the largest n-gram we have found running the dependency structure algorithm with *fringe heuristic off*, *number replacement on* and no query specified. Below the 47-gram is the corresponding sentence of the Penn Treebank (without number replacements). The Sentence occurs twice, in this case we would not need the *number replacement*. The parts where we found varying dependency structure is bold.

(3.18) 47-gram: *For each poll, the* ***odds are [NWC] out of [NWC] that*** *if pollsters had sought to survey every household in the U.S. using the same questionnaire, the findings would* ***differ*** *from these poll results* ***by no more than [NWC] [NWC] percentage points*** *in either direction*

Sentence as it is in the corpus (found twice): For each poll, the **odds are 19 out of 20 that** if pollsters had sought to survey every household in the U.S. using the same questionnaire, the findings would **differ** from these poll results **by no more than 2 1\/2 percentage points** in either direction

In total we found three different dependency structures within this sentence. Figure 3.6 shows the first. The dependency structure varies greatly for the shown parts 1a) and 1b).

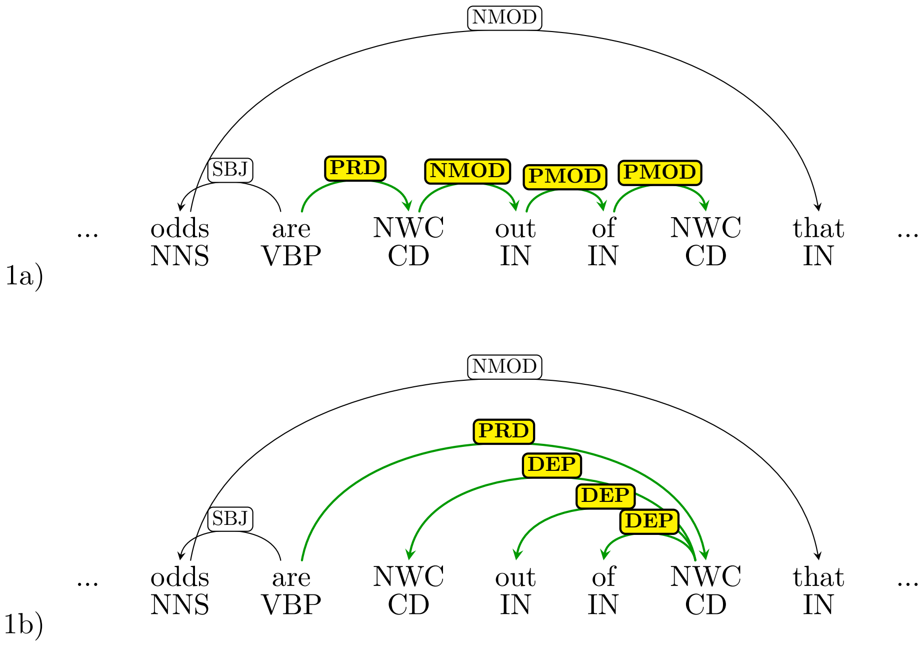


Figure 3.6: The figure shows the first of three varying dependency structures of the example sentence shown in Example 3.18. The relevant edges where variation occurs are coloured green. Below each word-form is the corresponding part-of-speech tag. NWC = numberwildcard

On the other side in Figure 3.7 2a) and 2b) we only found different dependency relation tags the edge direction is the same. The last varying part we found is shown in Figure 3.8 where we have pos-tag variation <1\/2, {CD, NN}> (coloured red) in addition to the dependency structure variation. The pos-variation was found during the pos error mining process as well.

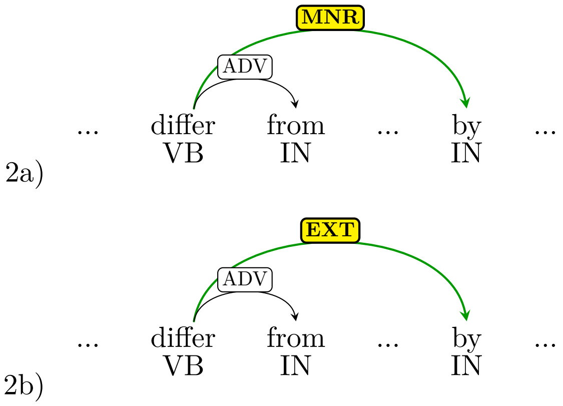


Figure 3.7: The figure shows the second of three varying dependency structures of the example sentence shown in Example 3.18. The relevant edges where variation occurs are coloured green. Below each word-form is the corresponding part-of-speech tag. NWC = numberwildcard

.

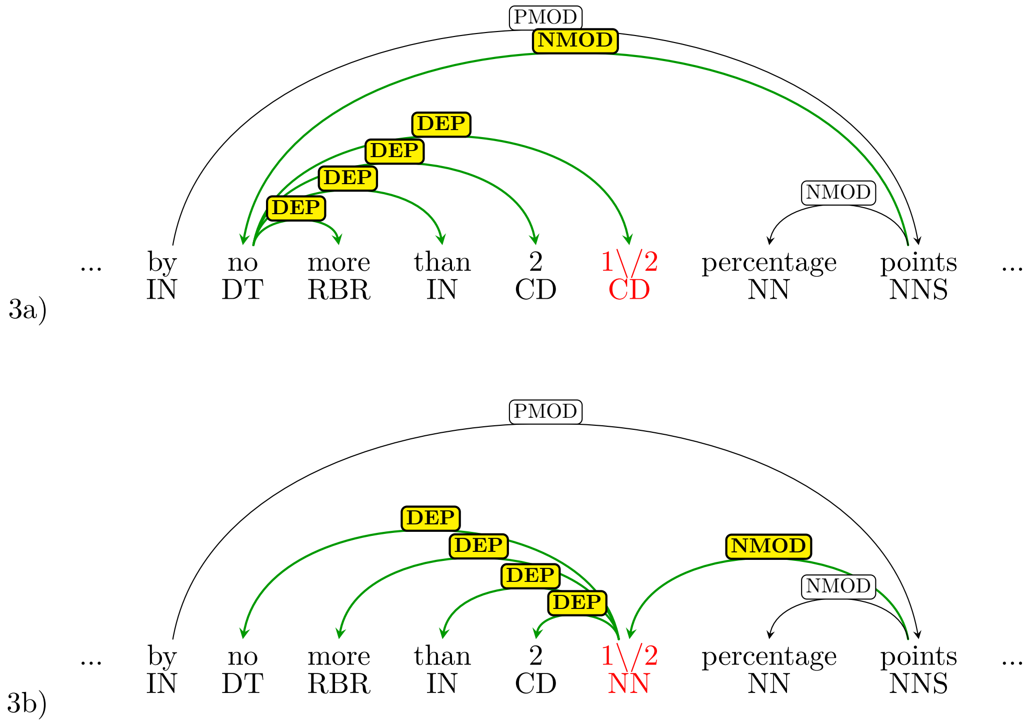

PMOD
NMOD
DEP
DEP
DEP
DEP
NMOD
... by no more than 2 1\/2 percentage points ...
IN DT RBR IN CD CD NN NNS
3a)
PMOD
DEP
DEP
DEP
DEP
NMOD
NMOD
... by no more than 2 1\/2 percentage points ...
IN DT RBR IN CD NN NN NNS
3b)

Figure 3.8: The figure shows third varying dependency structure of the Penn Treebank sentence shown in Example 3.18. Below each word-form is the corresponding part-of-speech tag. Contrasting to Figure 3.6 we discovered variation within the dependency structure (green) and additional within the pos-tags for <1\/2, CD, NN>

3.3 Conclusion

Comparing the algorithm passes and looking at which point we have the same result-set varies a lot. For the TIGER Corpus this occurs already in the 36-pass. A point where the algorithm still would do 19 more pass before reaching the final 56-gram. As a side effect the results we found mining TIGER do not differ for all *passes>36* no matter which parameters were enabled. This looks very different for the Penn Treebank where on one hand we get less results when we disable the number wild cards (55 instead of 64) and even the result sets collapse for all four options first when reaching the penultimate iteration step.

We have showed that using number wild cards is really a benefit during the error mining process since the algorithm cannot miss any sentences that should be treated equal but aren't because of different numbers within. The results we found in the Penn Treebank shows this very clearly, where we discovered longer n-grams when we turned *number replacement on.* We take this is a side effect of the data. The Penn Treebank training data we used is part of the Penn Treebank and the sentences originate from the Wall Street Journal, which includes a lot of numbers. On the other hand for the German TIGER there were no hits that we would have missed, but still the total count of different tags was decreased when *number replacement* was switched on. The number replacement seems to be a very useful option to have enabled when doing error mining.

An observation we have made using dependency structure error mining is, that erroneous pos-tagging seems to propagate. Sentences with wrong pos tags seem to have a wrong dependency structure. Nevertheless running the dependency algorithm also delivers erroneous sentences we would not find by pos error mining. We believe that a sentence which full-fills the criteria that there is both pos and dependency errors this be a strong indicator that there is something wrong and may be useful when ranking the errors.

The manual evaluation for the TIGER Corpus showed that the algorithm performed well. We were able to detect errors at a precision of at least 99.16% for 115 sampled 6-grams (118/119 variation nucleus are erroneous). Looking at n-grams with a size of 6 seems to be sufficient when detecting errors within the TIGER Corpus.

4 Related Work

As mentioned before, the algorithm of Dickinson and Meurers (2003) to detect errors using variation within n-grams is still state of the art. Since the algorithm is flexible it can be used for more than just pos error mining. For example for dependency structure error mining as introduced by Boyd et al. (2008) which we implemented as well. Besides that, there exists also a method for discontinuous structural annotation (Dickinson and Meurers, 2005). Like the pos error mining they try to find strings annotated for constituents. Afterwards they compare these strings to any other occurrences elsewhere whether annotated as a constituent or not. For this they developed a method to find strings which are not treated as constituents. These "non-constituents" were assigned with a special NIL category label. They never need to search for NIL strings across sentence boundaries since syntactic annotation itself respects them. For an efficient search they use a trie data structure which is generated in the first pass when processing the corpus. To detect candidates in the following step they check if there is a match for a given string within the tree (if there exists a path). If there exist a path for a given string inside the trie data structure. Finally they run a filter that eliminates NIL tagged strings that overlap with the identical constituent strings from consideration.

Dickinson (2007) shows a mechanism to determine ambiguity classes for pos-tags. He uses two different filters, first for classes which only occur very rarely. Besides that, he only replaces tags by ambiguity classes once they were flagged by an error detection phase as shown in MacKinlay and Baldwin (2005). To determine if an ambiguity class is typical they use a frequency-based criterion n, which is empirically determined. With this method they were able to pick words or ambiguities. They pointed out that grouping words together using their pos ambiguity classes is useful to improve pos tagging.
We adapted the idea of "re-tagging" in our query mechanism where we are able to replace tags during the error mining process. We do not group words using their pos-tag classes. If one tag class is modified the tag changes for all occurrences but no further grammar checks are made. The query editor is rather used to filter the error mining results than improving pos tagging.

Another error detection method for error detection and correction in dependency treebanks was introduced by Volokh and Neumann (2011). The authors use the graph-based MSTParser (McDonald et al., 2005) and transition-based MaltParser (Nivre et al., 2007) trying to get as close to the gold standard which they were trained on. The two parsers have completely different architectures hence they make different mistakes during the parsing process. The accuracy they reached was 98% and 99%. The authors pointed out that reaching 100% is impossible as far as some phenomena are either too rare that they can not be covered by parsing rules or the prediction was correct but the gold-standard was wrong. They try to find errors where both parsers deliver different dependency results compared to the gold-standard. After this first step they substitute the dependencies by using the results of the MSTParser.

Then they re-parse the resulting modified corpus with the transition-based MDParser. [1] The use of MSTParser output for substitution is to prevent one parser having advantage over another since MDParser and MaltParser are both transition-based, and more likely to make the same mistakes. The modification is kept when the MDParser predicts the same otherwise its undone. The author suggests that dependencies which were formerly wrong and became correct after running the three steps are likely errors within the gold-standard. This is based on the fact that two parsers deliver a result that differs from gold which is later approved by a third parser. To compare their results they counted the dependency occurrences $B \underset{parser}{\longrightarrow} A$ (B is head of A, predicted by the parser) and $C \underset{gold}{\longrightarrow} A$ (C head of A, proposed by the gold standard). They say to detect variation both counts must be available. In case the parser count outweighs the gold count this would be a strong indicator for erroneous tagging. They found out that this would only work in 39.5% for their 3535 classified errors. They pointed out that with their method they find different kind of errors compared to variation detection.

A GUI-based tool for error detecting was build by Agarwal et al. (2012b). It supports different types of error detection for hindi dependency treebanks and is integrated within the Sanchay tool introduced by Ambati et al. (2010b). Sanchay is an open source platform to work on languages and developing Natural Language Processing (NLP). Sanachay is able to detect errors on various levels e.g. pos, chunk, morphological and dependency level. After choosing a method the user can correct erroneous instances manually. Detected errors are represented in a table listing and clicking on a specific error instance shows the corresponding sentences with the matching node highlight to the user. Additionally the table contains information about the parent/sibling dependency labels and part-of-speech tags. The authors say that this is often already enough information for the annotators to correct errors without viewing the sentence. Our implementation only targets pos or dependency structure errors.

[1] http://mdparser.sb.dfki.de

5 Summary and Future Work

Our motivation was to develop a GUI-based tool to error-mine linguistically annotated corpora. Since corpora are used to evaluate automatic language processing systems its worthwhile to reduce the number of inconsistencies inside because this will improve the evaluation quality. The approach implementing state-of-the-art computational linguistics algorithms for error detection into a user-friendly interface increases the operation domain because the algorithms can be used by a wider audience without deeper knowledge in computers. We built a GUI-based tool to search for erroneous part-of-speech tags, using the algorithm shown in Dickinson and Meurers (2003) for part-of-speech tags. To detect errors within dependency structures we implemented the method introduced by Boyd et al. (2008). With the resulting system the user is able to run the error mining algorithm with only a few mouse-clicks. The resulting n-grams are represented graphically, providing an easy way to manually review the results.

As we showed in Chapter 3 finding errors systematically helps determining erroneously tagged words within a given corpus. We evaluated our tool on the German TIGER Corpus and the English Penn Treebank. This was done using all possible error mining search parameters (fringe on/off, number replacement on/off). We showed in Section 3.2.1 that for the German treebank it does not matter which configuration was chosen since all were able to find the longest n-gram. This changed for the Penn Treebank where we are no longer able to find the same n-gram length with disabled *number replacement*. We believe this to be a side effect since the Penn Treebank contains a lot of sentences of the Wall Street Journal and so the number replacement tends to be especially useful when comparing data with a lot of numbers. To evaluate the dependency structures we switched the *number replacement* option on/off. For both treebanks we were able to detect sentences that we have found before during the pos error mining. They contain varying dependency structure and pos tags. Nevertheless there were sentences which were not related to the sentences we found during the pos error mining at all. Here nodes had the same pos tags but the dependency structure varies therefore they only occur in the dependency structure result set.

We performed a manual evaluation where we sampled 115 6-grams (fringe heuristic on, number replacement on). For 94.96% we can determine the correct tag. For 4.20% of these we only can say that these were errors but determining the correct tag was too difficult because the annotation guidelines did not cover them. Nevertheless we also found one false positive: both tags were correct for the corresponding sentences. This happened because the current fringe heuristic implementation only requires one word at each side of the nucleus. So as a future work the fringe heuristic may be changed in a way that the user can specify how many words must be left/right of a nucleus. The total precision we get is 99.16%.

So far we cannot directly correct annotation errors. Generally, it is possible to turn ICARUS into an annotation tool. The error-mining plug-in provides the user with a graphical exploring mechanism to find complex linguistic phenomena inside the target corpus that could be useful to detect difficult tag decisions and therefore may help improving tagging guidelines.
However a future work could be to write an extension to annotate and correct corpora. To assist the annotator this could be improved by showing all tags (for one nucleus) that have been found within the selected n-gram. This approach may help improve consistency during the correction process and prevent further inconsistencies through a correction process.

As we have shown during the evaluation for dependency structure mining we detect sentences that we already found when using part-of-speech error mining (same target corpus). It seems that some errors occur in both subsets. It might be useful to build another query filter targeting this phenomenon. For example running the part-of-speech error mining algorithm followed up by the dependency structure algorithm. Of course as we showed during the dependency evaluation we can not assume that all sentences occurring during pos-mining match with the one found running dependency structure mining since there are inconsistencies that only show up in one of them. Nevertheless a second “confirm” pass might be useful to give a higher error ranking to those sentences that we detect in both algorithms. This could be useful when the user starts manually reviewing the results.

A Appendix

A.1 The STTS-Tagset

Table A.1: STTS Tagset as seen in Schiller et al. (1999)

Part-Of-Speech	**Description**	**Example**
ADJA	attributives Adjektiv	[das] große [Haus]
ADJD	adverbiales oder prädikatives Adjektiv	[er fährt] schnell, [er ist] schnell
ADV	Adverb	schon, bald, doch
APPR	Präposition; Zirkumposition links	in [der Stadt], ohne [mich]
APPRART	Präposition mit Artikel	im [Haus], zur [Sache]
APPO	Postposition	[ihm] zufolge, [der Sache] wegen
APZR	Zirkumposition rechts	[von jetzt] an
ART	bestimmter oder unbestimmter Artikel	der, die, das, ein, eine
CARD	Kardinalzahl	zwei [Männer], [im Jahre] 1994
FM	Fremdsprachliches Material	[Er hat das mit "] A big fish [" übersetzt]
ITJ	Interjektion	mhm, ach, tja
KOUI	unterordnende Konjunktion mit "zu" und Infitiv	um [zu leben], anstatt [zu fragen]
KOUS	unterordnende Konjunktion mit Satz	weil, daß, damit, wenn, ob
KON	nebenordnende Konjunktion	und, oder, aber
KOKOM	Vergleichspartikel, ohne Satz	als, wie
NN	Appellativa	Tisch, Herr, [das] Reisen
NE	Eigennamen	Hans, Hamburg, HSV
PDS	substituierendes Demonstrativpronom	dieser, jener
PDAT	attribuierendes Demonstrativpronomen	jener [Mensch]
PIS	substituierendes Indefinitpronomen	keiner, viele, man, niemand
PIAT	attribuierendes Indefinitpronomen ohne Determiner	kein [Mensch], irgendein [Glas]
PIDAT	attribuierendes Indefinitpronomen mit Determiner	[ein] wenig [Wasser], [die] beiden [Brüder]

Part-Of-Speech	**Description**	**Example**
PPER	irreflexives Personalpronomen	ich, er, ihm, mich, dir
PPOSS	substituierendes Possessivpronome	meins, deiner
PPOSAT	attribuierendes Possessivpronomen	mein [Buch], deine [Mutter]
PRELS	substituierendes Relativpronomen	[der Hund,] der
PRELAT	attribuierendes Relativpronomen, Relativpronomen	[der Mann ,] dessen [Hund]
PRF	reflexives Personalpronomen	sich, einander, dich, mir
PWS	substituierendes Interrogativpronomen	wer, was
PWAT	attribuierendes Interrogativpronomen	welche [Farbe], wessen [Hut]
PWAV	adverbiales Interrogativ- oder Relativpronomen	warum, wo, wann, worüber, wobei
PAV	Pronominaladverb	daür, dabei, deswegen, trotzdem
PTKZU	"zu" vor Infinitiv	zu [gehen]
PTKNEG	Negationspartikel	nicht
PTKVZ	abgetrennter Verbzusatz	[er kommt] an, [er fährt] rad
PTKANT	Antwortpartikel	ja, nein, danke, bitte
PTKA	Partikel bei Adjektiv oder Adverb	am [schönsten], zu [schnell]
TRUNC	Kompositions-Erstglied	An- [und Abreise]
VVFIN	finites Verb, voll	[du] gehst, [wir] kommen [an]
VVIMP	Imperativ, voll	komm [!]
VVINF	Infinitiv, voll	gehen, ankommen
VVIZU	Infinitiv mit "zu", voll	anzukommen, loszulassen
VVPP	Partizip Perfekt, voll	gegangen, angekommen
VAFIN	finites Verb, aux	[du] bist, [wir] werden
VAIMP	Imperativ, aux	sei [ruhig !]
VAINF	Infinitiv, aux	werden, sein
VAPP	Partizip Perfekt, aux	gewesen
VMFIN	finites Verb, modal dürfen	
VMINF	Infinitiv, modal	wollen
VMPP	Partizip Perfekt, modal	[er hat] gekonnt
XY	Nichtwort, Sonderzeichen enthaltend	D2XW3
$,	Komma	,
$.	Satzbeendende Interpunktion	. ? ! ; :
$(	sonstige Satzzeichen; satzintern	- [,]()

Deviation from STTS in the TIGER Corpus:

PIDAT vs. PIAT No distinction is made between PIAT and PIDAT. PIAT is used for indefinite pronouns in an attributive function both with and without determiner.

ADV Prepositions are tagged as ADV when the modify numerals.

PAV vs. PROAV The tag PROAV replaces the STTS tag PAV.

Table A.2: TIGER Corpus tagset for Node Labels shown in Smith (2003)

Node Tag	**Description**
AA	superlative phrase with *am*
AP	adjective phrase
AVP	adverbial phrase
CAC	coordinated adposition
CAP	coordinated adjective phrase
CAVP	coordinated adverbial phrase
CCP	coordinated complementiser
CH	chunk
CNP	coordinated noun phrase
CO	coordination
CPP	coordinated adpositional phrase
CS	coordinated sentence
CVP	coordinated verb phrase (non-finite)
CVZ	coordinated infinitive with *zu*
DL	discourse level constituent
ISU	idiosyncratic unit
MTA	multi-token adjective
NM	multi-token number
NP	noun phrase
PN	proper noun
PP	adpositional phrase
QL	quasi-language
S	sentence
VP	verb phrase (non-finite)
VZ	infinitive with *zu*

Table A.3: TIGER Corpus tagset for edge labels as showed in Smith (2003)

Edge Tag	**Description**
AC	adpositional case marker
ADC	adjective component
AG	genitive attribute
AMS	measure argument of adjective
APP	apposition
AVC	adverbial phrase component
CC	comparative complement
CD	coordinating conjunction
CJ	conjunct
CM	comparative conjunction
CP	complementizer
CVC	collocational verb construction (*Funktionsverbgefüge*)
DA	dative
DH	discourse-level head
DM	discourse marker
EP	expletive *es*
HD	head
JU	junctor
MNR	postnominal modifier
MO	modifier
NG	negation
NK	noun kernel element
NMC	numerical component
OA	accusative object
OA	second accusative object
OC	clausal object
OG	genitive object
OP	prepositional object
PAR	parenthetical element
PD	predicate
PG	phrasal genitive
PH	placeholder
PM	morphological particle
PNC	proper noun component
RC	relative clause
RE	repeated element
RS	reported speech
SB	subject

Edge Tag	Description
SBP	passivised subject (PP)
SP	subject or predicate
SVP	separable verb prefix
UC	unit component
VO	vocative

A.2 Penn Treebank Tagset

Table A.4: Penn Treebank II Part-of-Speech tagset as shown in Santorini (1990)

Node Tag	Description	Example
CC	conjunction, coordinating	and, or, but
CD	cardinal number	five, three, 13%
DT	determiner	the, a, these
EX	existential there	there were six boys
FW	foreign word	mais
IN	conjunction, subordinating or preposition	of, on, before, unless
JJ	adjective	nice, easy
JJR	adjective, comparative	nicer, easier
JJS	adjective, superlative	nicest, easiest
LS	list item marker	
MD	verb, modal auxillary	may, should
NN	noun, singular or mass	tiger, chair, laughter
NNS	noun, plural	tigers, chairs, insects
NNP	noun, proper singular	Germany, God, Alice
NNPS	noun, proper plural	we met two Christmases ago
PDT	predeterminer	both his children
PRP	pronoun, personal	me, you, it
PRP$	pronoun, possessive	my, your, our
RB	adverb	extremely, loudly, hard
RBR	adverb, comparative	better
RBS	adverb, superlative	best
RP	adverb, particle	about, off, up
SYM	symbol	%
TO	infinitival to	what to do?
UH	interjection	oh, oops, gosh
VB	verb, base form	think
VBZ	verb, 3rd person singular present	she thinks

Node Tag	Description	Example
VBP	verb, non-3rd person singular present	I think
VBD	Verb, past tense	they *thought*
VBN	verb, past participle	a sunken ship
VBG	verb, gerund or present participle	thinking is fun
WDT	wh-determiner	which, whatever, whichever
WP	wh-pronoun, personal	what, who, whom
WP$	wh-pronoun, possessive	whose, whosever
WRB	wh-adverb	where, when
.	punctuation mark, sentence closer	.;?*
,	punctuation mark, comma	,
:	punctuation mark, colon	:
(	contextual separator, left paren	(
)	contextual separator, right paren	)

Table A.5: Penn Treebank II chunk tagset

Node Tag	Description	Words	Example
NP	noun phrase	DT+RB+JJ+NN + PR	the strange bird
PP	prepositional phrase	TO+IN	in between
VP	verb phrase	RB+MD+VB	was looking
ADVP	adverb phrase	RB	also
ADJP	adjective phrase	CC+RB+JJ	warm and cosy
SBAR	subordinating conjunction	IN	whether or not
PRT	particle	RP	up the stairs
INTJ	interjection	UH	hello

Table A.6: Dependency Labels for the Penn Treebank as introduced by (Johansson, 2008, p.145-146)

Node Tag	Description
ADV	General adverbial
AMOD	Modifier of adjective or adverbial
APPO[1]	Apposition
BNF	Benefactor complement (“for”) in dative shift
CONJ	Between conjunction and second conjunct

[1] Only used in the CoNLL-2008 treebank

Node Tag	Description
COORD	Coordination
DEP	Unclassified
DIR	Adverbial of direction
DTV	Dative complement (“to”) in dative shift
EXT	Adverbial of extent
EXTR	Extraposed element in cleft
HMOD[1]	Between a head and a dependent inside a hyphenated word
HYPH[1]	Between a part of a hyphenated word and a following hyphen
IM	Between infinitive marker (“to”) and a verb
LGS	Logical subject of a passive verb
LOC	Locative adverbial or nominal modifier
MNR	Adverbial of manner
NAME[1]	Name-internal link
NMOD	Modifier of nominal
OBJ	Object
OPRD	Predicative part of a small clause
P	Punctuation
PMOD	Modifier of preposition
POSTHON[1]	Posthonorific modifier of nominal
PRD	Predicative complement
PRN	Parenthetical
PRP	Adverbial of purpose or reason
PRT	Between verb and particle
PUT	Prepositional phrase complement of the verb “put”
ROOT	Root
SBJ	Subject
SUB	Between subordinating conj. and subordinated clause
SUFFIX[1]	Between possessor and possessive suffix
TITLE[1]	Between name and title
TMP	Temporal adverbial or nominal modifier
VC	Verb chain
VOC	Vocative

A.3 Error Mining Results

A.3.1 TIGER Corpus Results

The tables use the following abbreviations fringe heuristic (F), number wild-card (NWC)

Table A.7: Result for the German TIGER Corpus showing the results of the four error mining configurations we used during the evaluation (fringe heuristic on, number wildcards on), (fringe heuristic off, number wildcards on), (fringe heuristic on, number wildcards off) and (fringe heuristic on, number wildcards off)

N-Gram	F; NWC	no F; NWC	F; no NWC	no F; no NWC
total	87423	87423	89383	89383
1	4059	4059	4148	4148
2	7004	7004	6870	6870
3	3434	3434	3093	3093
4	448	948	369	758
5	260	388	230	331
6	207	310	195	246
7	208	253	148	184
8	185	216	121	150
9	136	162	109	135
10	118	141	105	128
11	109	127	105	123
12	102	118	102	118
13	98	112	98	112
14	92	104	92	104
15	84	93	84	93
16	74	82	74	82
17	66	73	66	73
18	59	66	59	66
19	54	61	54	61
20	50	57	50	57
21	46	52	46	52
22	43	48	43	48
23	41	44	41	44
24	39	41	39	41
25	36	38	36	38
26	33	35	33	35
27	30	32	30	32
28	27	29	27	29
29	25	27	25	27

N-Gram	F; NWC	no F; NWC	F; no NWC	no F; no NWC
30	23	25	23	25
31	21	23	21	23
32	20	22	20	22
33	20	21	20	21
34	20	21	20	21
35	20	21	20	21
36	20	21	20	21
37	20	20	20	20
38	19	19	19	19
39	18	18	18	18
40	17	17	17	17
41	16	16	16	16
42	15	15	15	15
43	14	14	14	14
44	13	13	13	13
45	12	12	12	12
46	11	11	11	11
47	10	10	10	10
48	9	9	9	9
49	8	8	8	8
50	7	7	7	7
51	6	6	6	6
52	5	5	5	5
53	4	4	4	4
54	3	3	3	3
55	2	2	2	2
56	1	1	1	1
57	-	-	-	-

Table A.8: Result for TIGER Corpus using two different queries: NN, NE and NN, NE, XY both executed with the error mining configuration (fringe heuristic on, number wildcards on)

N-Gram	Hits (NN + NE)	Hits (NN + NE + XY)
total	87423	87423
1	1335	1347
2	1018	1023
3	474	476
4	169	169
5	159	159
6	140	140
7	151	151
8	130	130
9	82	82
10	63	63
11	52	52
12	44	44
13	40	40
14	35	35
15	29	29
16	23	23
17	19	19
18	15	15
19	12	12
20	10	10
21	8	8
22	8	8
23	8	8
24	7	7
25	6	6
26	5	5
27	4	4
28	3	3
29	2	2
30	1	1
31	-	-

A.3.2 Penn Treebank Results

The tables use the following abbreviations fringe heuristic (F), number wild-card (NWC). In the first row “total” are the total amount of word forms we have found.

Table A.9: Result for the Penn Treebank training data. Showing the results of the four error mining configurations we used during the evaluation (fringe heuristic on, number wildcards on), (fringe heuristic off, number wildcards on), (fringe heuristic on, number wildcards off) and (fringe heuristic on, number wildcards off)

N-Gram	F; NWC	no F; NWC	F; no NWC	no F; no NWC
total	35050	35050	39782	39782
1	5423	5423	5578	5578
2	11783	11783	11819	11819
3	7318	7318	6795	6795
4	1959	3845	1609	3124
5	1480	2456	1105	1744
6	1201	1795	840	1200
7	984	1368	681	902
8	822	1105	551	714
9	716	928	473	595
10	612	777	399	498
11	540	679	347	429
12	493	614	209	377
13	438	536	278	334
14	388	473	253	303
15	346	421	228	273
16	313	375	205	242
17	285	336	185	214
18	256	300	165	188
19	228	264	145	163
20	198	233	124	142
21	174	207	105	122
22	151	180	87	102
23	132	157	76	88
24	112	136	63	75
25	94	116	52	63
26	77	96	43	54
27	63	81	36	47
28	54	69	32	41
29	47	61	29	38
30	42	54	27	35
31	38	45	26	32

N-Gram	F; NWC	no F; NWC	F; no NWC	no F; no NWC
32	32	38	23	28
33	27	32	19	24
34	23	28	17	22
35	20	24	15	19
36	18	22	13	17
37	17	21	12	16
38	16	20	11	15
39	15	19	10	14
40	15	19	10	14
41	15	19	10	14
42	15	19	10	14
43	15	18	10	13
44	15	18	9	12
45	14	17	8	11
46	12	15	7	10
47	11	14	6	9
48	10	13	5	8
49	9	11	4	5
50	8	10	2	3
51	8	10	2	3
52	8	10	2	3
53	8	10	2	3
54	8	9	2	2
55	7	8	1	1
56	6	7	-	-
57	5	6		
58	4	5		
59	3	4		
60	2	3		
61	2	3		
62	2	3		
63	2	2		
64	1	1		
65	-	-		

N-Gram	V*
total	35050
1	3326
2	5474
3	2255
4	320
5	190
6	159
7	144
8	140
9	132
10	123
11	117
12	108
13	94
14	83
15	73
16	65
17	57
18	51
19	46
20	40
21	36
22	32
23	28
24	23
25	18
26	12
27	8
28	5
29	4
30	3
31	3
32	2
33	1
34	1
35	-

Table A.10: Result for the Penn Treebank training data. Query: VB, VBZ, VBP, VBD, VBN and VBG (abbreviated V*). (fringe heuristic on, number wildcards on)

Bibliography

Agarwal, R., Ambati, B. R., and Sharma, D. M. (2012a). A hybrid approach to error detection in a treebank and its impact on manual validation time. In *Proceedings of the 10th International workshop on Treebanks and Linguistics Theories*, Heiderberg, Germany. The 10th International workshop on Treebanks and Linguistics Theories. (Cited on page 21)

Agarwal, R., Ambati, B. R., and Singh, A. K. (2012b). A gui to detect and correct errors in hindi dependency treebank. In Chair), N. C. C., Choukri, K., Declerck, T., Dogan, M. U., Maegaard, B., Mariani, J., Odijk, J., and Piperidis, S., editors, *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey. European Language Resources Association (ELRA). (Cited on pages 21 and 54)

Agrawal, R. and Srikant, R. (1994). Fast algorithms for mining association rules in large databases. In Bocca, J. B., Jarke, M., and Zaniolo, C., editors, *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, pages 487–499. Morgan Kaufmann. (Cited on page 17)

Ambati, B. R., Agarwal, R., Gupta, M., Husain, S., and Sharma, D. M. (2011). Error detection for treebank validation. In *Proceedings of the 9th Workshop on Asian Language Resources*, pages 23–30, Chiang Mai, Thailand. Asian Federation of Natural Language Processing. (Cited on page 21)

Ambati, B. R., Gupta, M., Husain, S., and Sharma, D. M. (2010a). A high recall error identification tool for hindi treebank validation. In Chair), N. C. C., Choukri, K., Maegaard, B., Mariani, J., Odijk, J., Piperidis, S., Rosner, M., and Tapias, D., editors, *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta. European Language Resources Association (ELRA). (Cited on page 21)

Ambati, B. R., Gupta, M., Husain, S., and Sharma, D. M. (2010b). A high recall error identification tool for hindi treebank validation. In Calzolari, N., Choukri, K., Maegaard, B., Mariani, J., Odijk, J., Piperidis, S., Rosner, M., and Tapias, D., editors, *LREC*. European Language Resources Association. (Cited on page 54)

Boyd, A., Dickinson, M., and Meurers, D. (2008). On detecting errors in dependency treebanks. *Research on Language and Computation*, 6(2):113–137. (Cited on pages 3, 11, 12, 15, 21, 53 and 55)

Brants, S., Dipper, S., Eisenberg, P., Hansen, S., König, E., Lezius, W., Rohrer, C., Smith, G., and Uszkoreit, H. (2004). Tiger: Linguistic interpretation of a german corpus. (Cited on page 9)

Dickinson, M. (2005). *Error detection and correction in annotated corpora.* PhD thesis, The Ohio State University. (Cited on pages 11, 12, 13, 15, 17, 24, 30, 33 and 41)

Dickinson, M. (2007). Determining ambiguity classes for part-of-speech tagging. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing 2007 (RANLP-07)*, pages 167–172, Borovets, Bulgaria. (Cited on pages 30 and 53)

Dickinson, M. and Meurers, W. D. (2003). Detecting errors in part-of-speech annotation. In *Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics (EACL-03)*, pages 107–114, Budapest, Hungary. (Cited on pages 3, 10, 11, 14, 53 and 55)

Dickinson, M. and Meurers, W. D. (2005). Detecting errors in discontinuous structural annotation. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL-05)*, Ann Arbor, MI, USA. (Cited on pages 15 and 53)

Gärtner, M., Thiele, G., Seeker, W., Björkelund, A., and Kuhn, J. (2013). Icarus – an extensible graphical search tool for dependency treebanks. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, Sofia, Bulgaria. Association for Computational Linguistics. (Cited on page 12)

Geoffrey, L., Garside, R., and Bryant, M. (1994). Claws4: The tagging of the british national corpus. In *Proceedings of the 15th International Conference on Computational Linguistics (COLING-94)*, pages 622–628, Kyoto, Japan,. (Cited on page 9)

Johansson, R. (2008). *Dependency-based semantic analysis of natural-language text.* PhD thesis, Lund University. (Cited on page 62)

MacKinlay, A. and Baldwin, T. (2005). Pos tagging with a more informative tagset. In *Proceedings of the Australasian Language Technology Workshop 2005*, pages 40–48, Sydney, Australia. (Cited on page 53)

Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2):313–330. (Cited on page 9)

McDonald, R., Pereira, F., Ribarov, K., and Hajič, J. (2005). Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, HLT '05, pages 523–530, Stroudsburg, PA, USA. Association for Computational Linguistics. (Cited on page 53)

Mel'çuk and A., I. (1988). *Dependency syntax: theory and practice.* SUNY. inci. (Cited on page 21)

Nivre, J., Hall, J., Nilsson, J., Chanev, A., Eryigit, G., Kübler, S., Marinov, S., and Marsi, E. (2007). Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):95–135. (Cited on page 53)

Ratnaparkhi, A. (1996). A maximum entropy model part-of-speech tagger. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP-96)*, pages 133–141, Philadelphia, PA. (Cited on page 9)

Santorini, B. (1990). Part of speech tagging guidelines for the penn treebank project. (Cited on pages 21 and 61)

Schiller, A., Teufel, S., Stockert, C., and Thielen, C. (1999). Guidelines für das tagging deutscher textcorpora mit stts. (Cited on pages 21 and 57)

Smith, G. (2003). A brief introduction to the tiger treebank, version 1. (Cited on pages 59 and 60)

Tesnière, L. (1959). *Eléments de Syntaxe Structurale*. Klincksieck, Paris. (Cited on page 21)

van Halteren, H. (2000). The detection of inconsistency in manually tagged text. In *Proceedings of the COLING-2000 Workshop on Linguistically Interpreted Corpora*, pages 48–55, Centre Universitaire, Luxembourg. International Committee on Computational Linguistics. (Cited on page 21)

Volokh, A. and Neumann, G. (2011). Automatic detection and correction of errors in dependency treebanks. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 346–350, Portland, Oregon, USA. Association for Computational Linguistics. (Cited on pages 21 and 53)

All links were last followed on September 26, 2013.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature