Tagging and parsing a large corpus

Research report

Svavar Kjarrval Lúthersson

2010

B.Sc. in Computer Science

Author:	Svavar Kjarrval Lúthersson
National ID:	071183-2119
Instructor:	Hrafn Loftsson
Moderator:	Magnús Már Halldórsson

Tölvunarfræðideild
School of Computer Science

0. Abstract

0.1 Abstract in English

This report is a product of a research where we tried to use existing language processing tools on a larger collection of Icelandic sentences than they had faced before. We hit many barriers on the way due to software errors, limitations in the software and due to the corpus we worked with. Unfortunately we had to resort to sidestep some of the problems with hacks but it resulted in a large collection of tagged and parsed sentences. We also managed to produce information regarding the frequency of words which could enhance the precision of current language processing tools.

0.2 Abstract in Icelandic

Þessi skýrsla er afurð rannsóknar þar sem reynt er að beita núverandi máltæknitólum á stærra safn af íslenskum setningum en áður hefur verið farið út í. Við rákumst á ýmsar hindranir á leiðinni vegna hugbúnaðarvillna, takmarkana í hugbúnaðnum og vegna safnsins sem við unnum með. Því miður þurftum við að sneiða hjá vandamálunum með ýmsum krókaleiðum en það leiddi til þess að nú er tilbúið stórt safn af mörkuðum og þáttum setningum. Einnig söfnuðum við upplýsingum um tíðni orða sem gætu bætt nákvæmni máltæknitóla.

Table of Contents

1. Introduction

Language is very complex. Most people know how to express themselves in their mother tongue yet they have a hard time of providing a complete definition of its structure. An enormous amount of research has been carried out with the goal of mechanising the processing of natural languages. Is a certain word a verb, a noun or something else entirely? What is the significance of that word for the sentence structure?

Each language has its own set of rules and sometimes they are known and in others they are not. The best known way to attempt to find unknown rules is to analyse a collection of sentences (corpus) in attempt to find them. Despite that, not all research has the primary objective of discovering or verifying an existence of such a rule and such discoveries can be a by-product of such a research.

The purpose of this research is to tag and parse (see section 2.2) a corpus which can be utilised in future works. Taggers could benefit from the statistical information generated and language researchers will have access to a tagged and parsed corpus to learn more about grammar usage and discover linguistic anomalies.

In this report, we will use a tool called CorpusTagger[2] to tag and parse a large Icelandic corpus. To date, an Icelandic corpus of this size has not been tagged and parsed before. We will talk about how the tools process a corpus such as this one in section 3. In section 4 we will discuss the bugs and limitations we discovered in the process along with possibilities for future work. Finally, in section 5, we will report the conclusions we made as a result of this research.

2. Background

In this section we present the corpus and the tools this research is based on.

2.1 The Corpus

The research is carried out on a large Icelandic corpus called Íslenskur Orðasjóður[1] (IOS corpus) which, to date, contains more than 250 million words. The text in the corpus was extracted from the Internet archives of 2 libraries in Iceland and went through data cleaning, further described in the paper about the construction of the IOS corpus. The archives are accessible on the website www.vefsafn.is.

The corpus is a 1.5 gigabyte plain text file with no specific encoding (see section 4.1.1). Each line starts with an entry number and then the text of that entry, separated by a tab character. The entry numbers are in sequence throughout the corpus and are unique. Each entry can contain more than one sentence.

For illustration, this is the first entry in the corpus:

```
1350892	Arnar Grant fitnessmeistari sá um að setja saman matseðil og ráðgjöf til starfsfólks.
```

2.2 Tagging and parsing

Tagging is a method to represent the classes and features of words in a standardised form, called a tag. Parsing is the process of organising tagged sentences into sentence segments to determine their structures.

In the tagging process, each word will receive a tag based upon its word class and other features it holds. Each tag consists of a sequence of characters, each with a separate meaning based on their location within the series. For example, the first character denotes the word class, which could be a noun expressed with the letter 'n', adverbs and pronouns are marked with the letter 'a', and 'x' is for if the word class is unknown. The remaining characters are based on features, or lack of them, that the word class has. The tag for the word 'matseðil' (e. "menu") is 'nkeo' where it denotes, in order, that the word is a noun (n), is masculine (k), in singular form (e) and is in the accusative case (o).

Parsing is the process of denoting the sentence structure by adding mark-ups to a tagged sentence. An example of mark-ups are verb phrases and noun phrases. They also add information about where the subjects and objects are in the sentence.

2.3 The IceNLP toolkit

The IceNLP toolkit[8] is a collection of tools designed for NLP (Natural Language Processing) with Icelandic words, as the name suggests. It contains the tokeniser[3] (section 2.4), IceTagger[3] (section 2.5), IceParser[14] (section 2.12) and other useful tools which are used in the process of tagging and parsing Icelandic text. Other taggers, such as fnTBL[5], can utilise the tools in the toolkit to make use of the rule-based property of the algorithms.

2.4 The tokeniser

The tokeniser is a program within the IceNLP toolkit which ultimately has the function of preparing the corpus for the taggers. It does not simply format the output so it has one word per line, it has to determine when sentences start and end (unless it is forced to ignore it). The input could consist of sentences which share a paragraph so the tokeniser has to find out where one sentence ends and another one starts. The period could mark an end of sentence, be a part of an abbreviation or any other known functions inside it. After going through the tokeniser, the input it considered tokenised.

2.5 IceTagger

IceTagger is a Part-of-speech (PoS) tagger which was developed mainly with Icelandic text in mind. It bases its operations on linguistic rules which make it a powerful tagger when it comes to Icelandic text. When faced with unknown words, it utilises a tool called IceMorphy to predict the possible tags.

The tagger is based on building Icelandic rules into it to determine the tag of each word. It also uses heuristics for disambiguation instead of relying on a large set of rules. This results in higher accuracy than other taggers when working with Icelandic text[12].

2.6 TnT

The Trigrams'n'Tags (TnT)[4] tagger uses a second order Markov model and has a case where it handles unknown words. Since it is data-driven, unlike IceTagger, TnT needs to be trained before it can understand a language. It uses a statistical model to determine the correct tag of each word. When facing unknown words, it assigns them higher probability to be a certain class depending on their suffix. For example, the suffix 'inn' gives higher probability that the word is a noun based on many other nouns that have the same suffix.

TnT expects the input to be in the ISO-8859-1 character encoding since it does not handle UTF-8 like all the other taggers (in this research) do.

2.7 fnTBL

The fast TBL system (fnTBL) is based on three main ideas; to generate transformation-based rules, fast training and multi-task classification. The authors, when introducing TBL, noted some serious drawbacks in the basic TBL algorithm, which they intended to fix. The fnTBL program is designed to scale well with increasing corpus sizes while still retaining its previous learning power, reducing training time and reduced memory requirements.

2.8 MXPOST

MXPOST[6] is a statistical model which is based on a maximum entropy model and also uses other contextual approaches to predict the PoS tag. It bases the probability on the word themselves along with the surrounding words. Like the TnT tagger, it also tries to determine the tag of previously unknown words.

The tagger expects the input to be a single sentence in each line whereas the others require or support a one word per line input. Since the output is under the same restrictions, it has to be converted back so it conforms with the output of the other taggers.

2.9 Bidir

Bidir[10] is a tagger developed by M. Dredze and J. Wallenberg and is based on a bidirectional PoS tagging method[11]. The method is based on reiteration and raises or lowers the weight of each possible tag as it progresses along with the weights of other words within a certain distance.

The downside is that Bidir, despite being the reportedly one of the most accurate taggers, is still too slow to process a very large corpus. We estimated that Bidir would have taken too long to tag the IOS corpus according to our calculations (see section 3.2). Therefore, we did not have the time to use that tagger.

2.10 Tagger training

All the taggers were supplied by our instructor and they were previously trained on the Icelandic Frequency Dictionary (IFD) corpus[15]. The IFD corpus consists of roughly 590 thousand tokens and has about 700 different tags.

2.11 CombiTagger

CombiTagger is a tool which implements a voting process[7] to pick the tag which receives the most votes. It tries first to see if there is a single tag which has the most number of taggers behind it. If so, that tag is selected. If there is a conflict among two or more equally sized groups of taggers, the process resorts to a tie-breaking method. The group which includes the tagger with the highest priority wins the vote and that tag is selected. The priority is determined by their measured accuracy so that the most accurate one receives the highest priority, the second-most accurate one in 2nd place and so on.

Word	IceTagger	TnT	fnTBL	MXPOST	Chosen tag
Og	c	c	nken-s	nken-s	c
þá	aa	aa	aa	aa	aa

Table 1: Example of how the vote might go in CombiTagger

In our example in Table 1, which is based on real output, 'þá' (e. then) gets the tag 'aa' since it has all the taggers behind it which makes it the largest group to support that tag. However, the vote for 'Og' (e. and) is in doubt since it passes the first check and there are two equally large groups of taggers that support them. But since IceTagger has the highest priority, the tag 'c', which it voted for, is selected.

2.12 IceParser

IceParser is a finite-state parser which processes a PoS tagged file and "produces annotations of both [its] constituent structure and syntactic functions" [9]. In other words, it marks each sentence with grammatical tags.

In its output, IceParser uses curly braces to show syntactic functions and square brackets to show the phrase structure. Only the word class and grammatical case information are used for the parser since using any other information would reduce its effectiveness in finding grammatical errors.

2.13 CorpusTagger

The main utility we use is the CorpusTagger, which is a collection of tools used to push the corpus through the tagging and parsing process. It prepares the corpus for each tagger we use since each tagger requires the input to be in a certain format. The main tools we use with the CorpusTagger are thus: The tokeniser, which formats the text to the tagger's input format; IceTagger, TnT, fnTBL, MXPOST, which are the taggers that associate the text with tags; CombiTagger, which makes the final decision of which tags to use by utilising a voting method; and finally, IceParser, which parses the tagged text and adds sentence structure information.

2.13.1 Preparing the corpus

Before tagging, the corpus must undergo cleaning and formatting changes. The input is expected to be in UTF-8 and might need some adjustments, if applicable, before the tokenisation process. It is also very important to get rid of the entry number so it does not influence the tagging and parsing later on. After the tokenisation the output goes through various fixes, which include getting rid of extra white-space characters. The result is a single file with one word (token) in each line. Between every two consecutive sentences is a blank line.

After tokenising, the first entry of the IOS corpus would look like this:

```
Arnar
Grant
fitnessmeistari
sá
um
að
setja
saman
matseðil
og
ráðgjöf
til
starfsfólks
.
```

2.13.2 Tagging the corpus

After the tokenisation process, CorpusTagger executes each tagger in succession, each working with the tokenised corpus. A tagger is a program which labels each token in the corpus with a tag (see section 2.2).

When a tagger does not support the format in which the tokenised file is in, CorpusTagger prepares a version the tagger supports. After the tagger has been run, if applicable, CorpusTagger formats the output so it conforms with the output format of the other taggers.

This is the output of the first sentence of the IOS corpus according to IceTagger:

```
Arnar nken-m
Grant nken-m <UNKNOWN>
fitnessmeistari nken <UNKNOWN>
sá sfg3eþ
um ao
að cn
setja sng
saman aa
matseðil nkeo
og c
ráðgjöf nveo
til ae
starfsfólks nhee
. .
```

Note the extra tag <UNKNOWN> which is used to signify that IceTagger had to look to IceMorphy to predict the tag (see section 2.5). The extra tag is removed before the next stage.

2.13.3 The combination method

CorpusTagger utilises a tool called CombiTagger to achieve consensus amongst the taggers. The algorithm goes through each token and its corresponding tags and uses a simple voting method to solve any conflicts. The output of each tagger needs to be converted to the same tag-set to make the combination procedure easier.

The main problem with using a combination method is when too many taggers agree on the same wrong tag, which means the correct tag is outvoted by the wrong ones. This also happens when all the taggers disagree but a tagger which chooses the wrong tag has more priority than the one which was correct. Researchers therefore have to keep this in mind when choosing the combination method and the taggers they intend to use.

The first sentence of the IOS corpus, after CombiTagger, now looks like that:

```
Arnar   nken-s
Grant   nken-s
fitnessmeistari nken
sá      sfg3eþ
um      ao
að      cn
setja   sng
saman   aa
matseðil        nkeo
og      c
ráðgjöf nveo
til     ae
starfsfólks     nhee
.       .
```

Comparing to the IceTagger output in the previous section, one would think that IceTagger was outvoted on the first two words, but CorpusTagger actually converts the tag from 'nken-m' to 'nken-s' before it is processed in CombiTagger.

2.13.4 Parsing the corpus and the aftermath

Before parsing the corpus, IceParser requires CorpusTagger to reformat the input so it is one sentence per line. Then IceParser processes the file and produces an output file with added information regarding sentence structure (see section 2.2). Separate programs produce error files containing sentence segments which IceParser thinks contain grammar errors. Since said error files do not contain line numbers, they need to be added afterwards.

The first sentence of the IOS corpus is parsed and this is the end result:
{*SUBJ> [NPs [NP Arnar nken-s Grant nken-s NP] [NP fitnessmeistari nken NP] NPs] *SUBJ>} [VP sá sfg3eþ VP] [PP um ao [VPi að cn setja sng VPi] PP] [AdvP saman aa AdvP] [NPs [NP matseðil nkeo NP] [CP og c CP] [NP ráðgjöf nveo NP] NPs] [PP til ae [NP starfsfólks nhee NP] PP] . .

Each word keeps its tag and multiple words are grouped together to form the sentence structure. 'Arnar Grant fitnessmeistari' (e. Arnar Grant fitness champion) is the subject of the sentence etc. The corpus is now considered to be parsed.

3 Tagging and parsing

In this section, we will describe the process of tagging and parsing the corpus and the associated work.

3.1 The test run

Since we had no first-hand knowledge of the running time of the taggers when processing a task of this size and due to time constraints, we decided to do a test run to infer how much time the taggers would require to tag the whole corpus. The results would enable us to see which taggers we would be able to run. We took the first 100 thousand entries in the corpus as a sample, tokenised them and ran them through the tagging process. The run also provided us with the opportunity to discover any apparent problems which could cause complications later.

3.2 Test running the taggers and the results

The presuppositions we made, when calculating the estimated running time, were that all the taggers would go through the corpus entries in constant speed and that the time we would measure would be 1/146th of the total running time since the total number of entries is roughly 14.6 million. All the tests were done on a 3.0 GHz computer. The purpose was to get a rough estimate of the running time so no extreme measures were taken to ensure no interruptions by other processes.

The results were that IceTagger, TnT and fnTBL would take 22.20 hours, 2.84 hours and 9.03 hours, respectively. The more time consuming taggers were MXPOST and Bidir. MXPOST would take 8.62 days and Bidir crashed after a little more than eight hours due to insufficient memory. We estimated that Bidir would have taken almost 49 days to reach this point if it would take the whole corpus and if memory was not an issue. We therefore decided to leave Bidir out in this research. MXPOST, like Bidir, crashed due to memory limitations but the problem was alleviated by splitting the corpus into manageable pieces.

3.3 Preparing the full run of the taggers

It took us two runs of the taggers to tag the corpus due to bugs in the tokeniser and the corpus that were discovered after the first round of tagging. The bugs are further described in sections 4.1.1 and 4.1.2. We tried to solve or circumvent all bugs we had found, before the second round of tagging.

It would have been too much to utilise a single computer for the task so we employed a cluster maintained by the School of Computer Science in Reykjavík University. The cluster consisted of 12 computers, with one computer acting as the main node which stored the home directory and the only one with a link to the outside world. The other eleven were the nodes which performed the assigned tasks. Unfortunately, we only got permission to use ten of them. Each node consisted of two CPU cores, either 3.0 or 3.2 GHz, and had two gigabytes of main memory.

The test run was very useful in determining any apparent limitations in the taggers. We made a test run of 50 thousand sentences and determined that the taggers would be able to handle that amount.

The corpus went through several procedures to prepare it for tagging. First, the corpus was converted from the ISO-8859-1 (the presumed encoding) to UTF-8. Second, we replaced all control characters, except tabs and spaces, with spaces. The reason we replaced them by spaces instead of deleting them was to ensure that the process did not combine words. Thirdly, we cut the entry numbers off so they would not influence the results. And finally, an empty line was injected amongst all corpus entries to circumvent a bug (section 4.1.2).

The tokeniser went through two stages: The first was to tokenise the input with a so-called 'input format 3' which makes the tokeniser try to detect for itself where sentences end and another one starts. The output was in a format with one token per line. The output was then converted to one sentence per line. The last step was in preparation to split the sentences amongst CPU cores in the cluster.

CorpusTagger was not designed to be run on a cluster so we needed to make a few adjustments. The corpus was split into 312 parts, each containing 50 thousand sentences, which were then assigned to each core in the cluster. A template from CorpusTagger was used to make a batch of commands to be executed on the cluster. A test run was executed with a single part of the corpus to make sure it would go through it without a hitch.

Each node in the cluster would go through each file in a certain directory, inferred from the parameters, consisting of the assigned corpus parts and executing the batched commands on each of them. Finally returning the results of each tagged output to directory on the main node. Then it would only be a matter of collecting the outputs.

3.4 The full run

Executing the full run went really well and the whole process ran its course in 13 to 14 hours which was in conjunction with our estimation inferred from the time measurements carried out before.

3.5 Running CombiTagger

CombiTagger is a memory hog because it needs to load all the tagger inputs at once. The tagger inputs were, on average, 3 gigabytes in size so more than 12 gigabytes of main memory would be needed if the whole thing had to be processed in one go. It was therefore essential to process each part of the corpus independently in CombiTagger before joining them together again. This limitation is described in section 4.2.2.

3.6 Running IceParser

The run of IceParser was uneventful bar one problem: It consumes a large quantity of hard drive space. This is further described in section 4.2.3. It was solved by deleting files as soon as they were not needed. The final output consisted of a parsed corpus with a sentence in each line not unlike the sample in section 2.13.4.

3.7 Connecting the results to their entry number.

In our view, it was very important to reattach the parsed corpus entries to their original entries in the corpus. Since we had to cut loose the corpus entry numbers, we had to resort to other methods to trace them back their origins and reattach them. This was not as simple as a 1-on-1 comparison since each corpus entry could contain multiple sentences.

The storage method chosen for this task was a database with two tables. The main table would store the corpus entry number along with its associated text. The other table would store three columns: The first one containing the number of the corpus entry, the second one its place within the original entry (sentence no #) and the third one the output as it came out of IceParser.

In order to be able to trace the parsed sentence to its original entry, one would need to be able to compare the corpus entry to the output. Since, by design, the CombiTagger output should have an equal number of sentences, it is sufficient to compare the corpus entry to the sentence according to CombiTagger instead of the IceParser output. To reduce complexity, we removed the tag from the CombiTagger output and reformatted the file so it contained a sentence per line. We compared the cleaned corpus, with the entry numbers, to the CombiTagger 'sentence per line' output. The reason we did not utilise the tokenised file with a sentence per line is discussed in section 4.1.2 and the reason we did not compare to the IceParser output itself is discussed in section 4.1.3.

Pseudo-code for matching the entries and inserting into a database:

```
---
for sentence in cleaned corpus
    strip corpus sentence of spaces
    len_c = length of stripped corpus sentence
    while len_c is larger than 0
            retrieve CombiTagger sentence
            strip CombiTagger sentence of spaces
            len_t = length of stripped CombiTagger sentence
            len_c = len_c - len_t
            insert CombiTagger sentence into db
    insert corpus sentence into db
---
```

The comparison process uses four files, two of which are the entries in which we will inject into the database and the other two are used to link them together. The cleaned corpus file is iterated line by line and in each iteration we iterate through the CombiTagger file until we have retrieved all sentences which belong to the corpus entry in question.

When processing a corpus entry, each entry is stripped of spaces so, in essence, it only contains the actual words and symbols it represents without any spaces. This is done due to the injection of extra spaces when tokenising the corpus. The same is done to the line in the CombiTagger 'sentence per line' file (the comparison entry). Now we know how long each entry consumes. If the corpus entry is longer than the comparison entry, we can be certain that there is at least one more entry we need to compare to. We subtract the length of the comparison entry from the corpus one, insert the IceParser entry into the database with its connection to the corpus entry, retrieve the next comparison entry and repeat the process until we have matched all sentences in that entry. As an extra precaution, we also check that the actual contents of the comparison entry is a sub-string of the corpus entry.

3.8 Statistical information for taggers

Taggers, especially statistical taggers, assign probabilities (weights) to each tag and use them to decide what tag it recommends for each token. Some of that information is based on the frequency of how often the token is associated with each tag. To generate this information, we ran a script on the CombiTagger output, mainly through a series of system commands commonly used in UNIX-based systems: sort CombiTagger_file | uniq -c | sort -nr > CombiTagger_file.freq

The first command sorts the lines of the CombiTagger output so each token and its associated tag are together with other instances of that word and tag. Its output is piped to the 'uniq' command which combines lines which are exact matches with the corresponding number of instances in front. The output is again piped to the sort command which sorts it in descending order. The results of all that is a list sorted by the number of instances, the token itself and then the tag, all in descending order.

The first 10 lines of the output:

```
15643359
13638268    .     .
9447129     og    c
7111082     ,     ,
6079844     að    cn
4987919     í     aþ
4905340     er    sfg3en
3803822     á     aþ
3127895     sem   ct
2986325     að    c
```

The number in front is the total number of instances in which the token appeared, the second one the token itself and finally the associated tag. The top line in our example is empty because it counts empty lines which corresponds to the number of sentences.

Using the input from the previous stages, the final stage consists of running a script which generates a frequency file stating each token and the number of appearances for each tag.

Examples of lines from the frequency file:

```
Fé 352 nhen 291 nken-s 30 nheo 20 nheþ 5 nkee-s 3 nkeo-s 2 nkeþ-s 1
Fèkk 5 sfg3eþ 2 nven-s 2 lvensf 1
Fèlagi 1 nheþ-s 1
```

3.9 The Morphological Database of Icelandic Inflections (MDII)

A database has been maintained by Árni Magnússon Institute of Icelandic Studies, containing a large database of Icelandic inflections. It was made available to the public in November 2009 but under a limited license. This part of the research was carried out mainly to see how widely it covered the inflections in the corpus and to help enhance its coverage by providing the most-used inflections currently not contained in the database.

The database was retrieved from MDII's website[13] and the inflections were inserted into the database. A script was made which went through the list of instances, as described in section 3.8, and queried the database to see which of them were there. Every inflection not found in the database was written to a file.

Note that the purpose of the database is to cover inflections so word categories like conjunctions, adverbs, prepositions and abbreviations cannot be found there. We can also exclude numbers and other symbols like periods, commas, exclamations marks and so on. Here are the top ten inflections which conform with the previously mentioned conditions, starting with its frequency, then the word and finally its tag. We also included the top 'normal' noun and adjective.

185210	hvað	fshen
77990	hvað	fsheo
60635	Hvað	fshen
43164	hverjum	fokeþ
40362	hverju	foheþ
40242	hverju	fsheþ
39433	eg	fp1en
38943	konar	nkee
35670	hvert	foheo
30869	ad	nhen
...		
5065	Spjallþráður	nken-s
3962	fleirum	lkfþsm
2463	fleiru	lheþvm
...		

The top six words and the ninth one are all inflections derived from the lemma 'hver' (e. who), an interrogative pronoun. The lemma for 'hver' does exist in the database but only as a noun with the meaning of 'geyser'. The word 'eg' is an older usage of the word 'ég' (e. I), mostly used in the old Icelandic although people tend to use it when their keyboard settings are not configured to type Icelandic text. People also tend to write 'ad' instead of 'að' due to keyboard configurations but its frequency could also be influenced due to Latin phrases as some of the text is 'law speak'. However, the word 'að' does not have any inflections and seems to be incorrectly categorised as a noun. The word 'konar' is not a normal noun as it does not morph.

Browsing the top entries of the most frequent words not in MDII, we noticed a lot of wrong spelling, either showing that MDII's coverage is extensive or some misspellings are too popular. It was not until we went down to words used less than 5500 times that we found useful words not in the database. One useful noun was 'Spjallþráður' (e. forum thread). We also found two inflections of adjectives which, by further checking, are in the database but all inflections in the comparative are listed as 'fleiri' (e. more), which does not seem to add up.

4. Discussion

In this section, we will analyse the results and report the bugs and limitations we came across. In summary, we found various software bugs and limitations which could have been avoided with more test cases and development time. There were also bugs or faults in the corpus which could have been avoided with more data filtering. We will also discuss possible future work which can be based on this research.

4.1 Bugs

In this section, we will discuss bugs we found in the software and in the corpus.

4.1.1 The corpus

The corpus was very useful in the research but it had its flaws. The corpus underwent some cleaning, as described in its paper but inherent errors were still to be found. The purpose is not to cast blame on the researchers who prepared the corpus, but to help them learn from their mistakes so better corpora can be generated in the future.

One flaw was its character encoding, ISO-8859-1, which limits the range of characters the corpus can represent. That was at least the detected encoding. With so many variations of character encodings, an encoding which has a union of all other encodings should have been chosen for the task. Unicode has the most potential as it covers the most number of characters. We presumed all the sentences were in ISO-8859-1 at the start of the project. But we came across some sentences in the corpus which were in 'double-Unicode' as they produced sentences which were obviously in Unicode before we converted them in the first place. To simplify any future sentence collecting for corpora, the sentences should be converted to Unicode during the extraction because it is more likely that the encoding information is available at that stage.

Examples of a corpus entries which did not work out due to character encoding issues:
2105703 BJBJ14.jpg\\ \\:\\:\\ description=F\\u00F3r tvisvar \\u00E1 sk\\u00ED\\u00F0i til Arosa \\u00ED Sviss me\\u00F0 vini m\\u00EDnum Magn\\u00FAsi heitnum Gu\\u00F0mundssyni bl\\u00F3masala og t\\u00F3k hann \\u00FEessa mynd af m\\u00E9r \\u00FEar.
2105714 \\u00CD f\\u00F6r me\\u00F0 m\\u00E9r voru \\u00FEeir Haraldur Johannessen r\\u00EDkisl\\u00F6greglustj\\u00F3ri og \\u00DEorsteinn Dav\\u00ED\\u00F0sson a\\u00F0sto\\u00F0arma\\u00F0ur minn.

In our attempt to convert the corpus to UTF-8, we noticed Unicode control characters which can only be found in Unicode. One of them is character 0x0085 (Next Line). We have not researched exactly why but for some reason, Icelandic quotation marks are classified as control characters in Unicode. Until the taggers can handle them better, it is better to convert them to English quotation marks (Unicode character 0x0022) which are currently not classified as such.

It is our observation that the IOS corpus might need more cleaning. Some lines were 'nonsensical' as they did not contain any sentences, just some garbage. This is, however, understandable since we cannot expect the authors to go over the corpus entries by hand or be successful in developing a method which can eliminate all such cases. One of the filtering method chosen was to see if any of the most frequent words in Icelandic appeared in the sentence and if any of them did, the entry would be classified as Icelandic. It seems like the implementation needs some more work.

Examples of an entries which contained no useful sentences:
1672493 w+wdl:xa :eA=E \\X #-"?
2274857 page_id=' + page_id; the_html += '\\t ' + page_name + ' \\n' the_html += '\\n'; if (y!

Finally, there are general concerns regarding the corpus, which are not wholly blamed on the authors, but rather on the HTML coding practices. Some entries were made up of answer choices and then the question, showing that the HTML 'input' tags were removed and the sentences left as they were, resulting in the following entry:
1350903 Já Nei Er eitthvað sem mælir á móti því að þú vinnir við matvæli t.d. óþol?

As stated before, this cannot be wholly blamed on those who prepared the corpus because such precise extraction would take too much time and not be of much gain considering the effort. This is however a factor to take note of when considering the results.

4.1.2 The tokeniser

The most serious bug was in the tokeniser which would meld corpus entries with each another. It hampered our efforts to link the sentences to their corresponding corpus entry. To save time and due to the complexity of the bug, we decided to tag the corpus again. After further research, we discovered that this bug was circumvented by injecting an extra newline among all corpus entries before they are tokenised.

Another bug we encountered was lack of support for special Unicode characters. It is our observation that any unknown character is classed as a separate word, causing great havoc when multiple unknown characters are in sequence. The measure taken was to filter as many of them out as possible.

After tagging the corpus on the cluster, we noticed another problem, the number of sentences did not correspond to the number of sentences we sent in. When making the tokeniser detect the start and end of sentences (input format 3), as discussed in section 3.3, it skipped some sentence markers. When the tokeniser was executed again with input format 2, forcing it to think there is exactly one sentence in the line, we noticed it disobeyed the command and still split them further into more sentences as it saw fit. This forced us to use the CombiTagger output when tracing the parsed sentences to their origins.

Since the text is from websites, a special concern came into light when we discovered that there is a special character for non-breaking spaces (0x00A0). The tokeniser did not recognise them as a space character and so classified them as separate tokens. This was not discovered until after CombiTagger finished. The taggers assigned tags to it so the tag assignments of nearby words could be skewed where non-breaking spaces were encountered. We also expect this problem to spread further and cause skewed results in the parsing process as well.

4.2.3 IceParser

In attempt to load the parsed the data into a database, we tried to use the parsed data by filtering out the IceParser syntax and every second word, starting from the first, would be from the corpus. Unfortunately, we discovered that tags intermittently get lost in the parsing process. The bug seems to be rare but should be noted and fixed as soon as possible.

4.2 Limitations

In this section, we will the discuss software limitations we encountered. Some of them were due to software design.

4.2.1 MXPOST

As mentioned in section 3.2, MXPOST has a long running time compared to the other taggers. This could discourage researchers from using it for large corpora. It also consumes more memory which makes it more hazardous to use than the others, especially if it crashes late in the tagging process. Here, we had to spend a considerable time trying to divide the corpus into small enough pieces for it to handle and to be sure it would be able to finish them without crashing before executing the batched commands discussed in section 3.3.

4.2.2 CombiTagger

As discussed in section 3.5, CombiTagger consumes too much memory. The reason is a design decision where the authors intended the program to be run behind a graphical user interface where the user is supposed to be able to view the data. This is obviously something which should be fixed in a later version. Circumventing this limitation required feeding it small enough pieces at a time.

4.2.3 IceParser

Due to its nature of being a state machine, it goes through each sentence segmentation type at a time, producing an ever-growing file output each time. With such a large corpus it can be a problem when hard disk space is limited. Because each stage is based on the stage before, the solution was to delete files when they were no longer needed. It is therefore our recommendation that such a procedure to be formally carried out in IceParser which can be turned off (i.e. to not delete files) by entering a so-called debug mode.

4.2.4 PPNPVP line numbers

Three separate programs run after IceParser in attempt to find suspected grammar errors and attach line numbers to errors that IceParser has output before. The error output was expected to be useful in effort to see if and where it makes mistakes. The first one takes care of prepositional phrases (PP), the second one noun phrases (NP) and the third one verb phrases (VP), hence the name PPNPVP. In executing the process of attaching line numbers, the process loads the CombiTagger output and the IceParser output, consuming a lot of memory in the process. This can be fixed by carrying out another method or simply integrate the functionality into IceParser.

5. Future work, Acknowledgements & Conclusion

5.1 Conclusion

Many obstacles have been overcome during this research, mostly due to unpredictability of the task: A corpus format the tools does not ordinarily deal with, bugs which are not normally encountered and reaching limits not normally reached. When starting the research, we only saw the surface of the task and had no way of knowing how it would turn out.

We discovered bugs, or rather faults, in the corpus which can be traced to insufficient cleaning and formatting. But would it be right to blame everything on those who prepared the corpus? Maybe the fault lies with the authors of the tools for not expecting the states the text would be in. Whoever is at fault, the bugs need to fixed.

Understandably, there are certain limits that can be reached when handling any task. The most hit barrier is memory and the second one is CPU processing power. Both of which increase with time, enabling us to handle bigger and bigger tasks. Yet, there is still room for improvement as some of the programs hit the barrier sooner than necessary due to design faults. We can only hope that the authors of the tools will take these issues under consideration as soon as possible.

The Morphological Database of Icelandic Inflections seems, based on this research, to have a good coverage although it might need some fixes here and there. Despite being a database of inflections, it might benefit from inclusions of adverbs and pronouns for completeness.

In conclusion, we can be fairly sure that we have only scratched the surface in natural language processing and research in this field is still going on because we have a lot left to do. In the scientific world, there is no complete certainty. There is however one exception, we can be certain that the search for facts is never complete.

5.2 Future work

Unfortunately, the bugs and limitations were in greater numbers and complexity than expected so we did not have time to do all the things we wanted to do. It is our hope that other researchers will use what we produced and continue our work. The parsed corpus can be used to measure the accuracy of the taggers and the parser in effort to enhance the process. It can also be used to produce information about verb sub-categorisation frames and the frequency information can potentially be used to enhance the accuracy of other taggers that are able to utilise the information.

5.3 Acknowledgements

Special thanks go to our instructor, Hrafn Loftsson, for being a great teacher and being helpful during time of need. Without his help, the research would not have gone as far as it did. We also want to thank the School of Computer Science at the University of Reykjavík for giving us access to the facilities we needed to work on this research and for granting us permission to use one of their computer clusters.

Other thanks go to Magnús Már Halldórsson, Björn Þór Jónsson and Hannes Högni Vilhjálmsson for sitting the status meetings and giving good advice. And finally, thanks to everybody else who has helped with the project, directly and indirectly.

References

[1] E. Hallsteinsdóttir, T. Eckart, C. Biermann, U. Quasthoff. and M. Richter. 2007. Íslenskur orðasjóður – Building a Large Icelandic Corpus. In Proceedings of the 16th Nordic Conference of Computational Linguistics (NoDaLiDa 2007), Tartu, Estonia.

[2] H. Loftsson, J. H. Yngvason, S. Helgadóttir and E. Rögnvaldsson. 2010. Developing a POS-tagged corpus using existing tools. In *Proceedings of "Creation and use of basic lexical resources for less-resourced languages", workshop at the 7th International Conference on Language Resources and Evaluation, LREC 2010*. Valetta, Malta.

[3] H. Loftsson. 2008. Tagging Icelandic Text: A linguistic rule-based approach. *Nordic Journal of Linguistics, 31(1), pages 47-72*.

[4] T. Brants. 2000. TnT - A Statistical Part-of-Speech Tagger. In *Proceedings of the 6th Applied NLP Conference, ANLP-2000*, Seattle, USA.

[5] G. Ngai and R. Florian. 2001. Transformation-based learning in the fast lane. In *Proceedings of NAACL-2001*, Carnegie Mellon University, Pittsburgh, USA.

[6] A. Ratnaparkhi. 1996. A Maximum Entropy Model for Part-Of-Speech Tagging. In *Proceedings of the Empirical Methods in Natural Language Processing Conference*, University of Pennsylvania, Philadelphia, USA.

[7] V. Henrich, T. Reuter and H. Loftsson. 2009. CombiTagger: A System for Developing Combined Taggers. In *Proceedings of the 22nd International FLAIRS Conference, Special Track: "Applied Natural Language Processing".* Sanibel Island, Florida, USA.

[8] H. Loftsson and E. Rögnvaldsson. 2007. IceNLP: A Natural Language Processing Toolkit for Icelandic. In *Proceedings of Interspeech 2007, Special Session: "Speech and language technology for less-resourced languages"*, Antwerp, Belgium.

[9] H. Loftsson. 2009. Correcting a PoS-tagged corpus using three complementary methods. In *Proceedings of the 12th Conference of the European Chapter of the ACI., pages 523-531, Athens, Greece, 30 March - 3 April 2009*.

[10] M. Dredze and J. Wallenberg. 2008. Icelandic Data Driven Part of Speech Tagging. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, Columbus, OH, USA.

[11] L. Shen, G. Satta, and A. K. Joshi. 2007. Guided learning for bidirectional sequence classification. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 760-767, Prague, Czech Republic, June 2007.

[12] H. Loftsson, I. Kramarczyk, S. Helgadóttir and E. Rögnvaldsson. Improving the PoS tagging accuracy of Icelandic text. In *Proceedings of the 17th Nordic Conference of Computational Linguistics (NoDaLiDa 2009)*. Odense, Denmark.

[13] http://bin.arnastofnun.is. Retrieved on the 8th of May 2010.

[14] H. Loftsson and E. Rögnvaldsson. 2007. IceParser: An Incremental Finite-State Parser for Icelandic. In *Proceedings of the 16th Nordic Conference of Computational Linguistics (NoDaLiDa 2007)*, Tartu, Estonia.

[15] J. Pind, F. Magnússon and S. Briem. 1991. *Íslensk orðtíðnibók [The Icelandic Frequency Dictionary]*. Reykjavík: The Institute of Lexicography, University of Iceland.