Practical Lab Session: MaltParser

The purpose of this practical lab session is to get acquainted with the MaltParser system by training and evaluating a dependency parser for a language of your choice. In addition, you will perform a simple error analys with respect to multiword expressions.

NEW: If you want to download the UD data sets that have been prepared for use with MaltParser, they are contained in UD_Data+MaltEval.tar.gz. As the name indicates, this archive also includes the MaltEval jar file for evaluation and visualization.

1. Install MaltParser

Go to the MaltParser website at maltparser.org. Go to the Download page and download the latest version in `maltparser-1.8.1.tar.gz`. Open a terminal and unpack the compressed archive by running:

```
tar xvf maltparser-1.8.1.tar.gz
```

Then change directory:

```
cd maltparser-1.8.1
```

Then run MaltParser:

```
java -jar maltparser-1.8.1.jar
```

This should produce output like the following:

```
----------------------------------------------------------------------------
                          MaltParser 1.8.1
----------------------------------------------------------------------------
         MALT (Models and Algorithms for Language Technology) Group
             Vaxjo University and Uppsala University
                             Sweden
----------------------------------------------------------------------------

Usage:
   java -jar maltparser-1.8.1.jar -f
   java -jar maltparser-1.8.1.jar -h for more help and options

help                   (  -h) : Show options
----------------------------------------------------------------------------
option_file            (  -f) : Path to option file
----------------------------------------------------------------------------
verbosity             *(  -v) : Verbosity level
  debug       - Logging of debugging messages
  error       - Logging of error events
  fatal       - Logging of very severe error events
  info        - Logging of informational messages
  off         - Logging turned off
  warn        - Logging of harmful situations
----------------------------------------------------------------------------

Documentation: docs/index.html
```

If you don't get this output, notify the the lab instructor.

2. Get some data

We are going to use treebanks from the first release of the Universal Dependencies project, which have been downloaded and prepared in 10 subdirectories of the directory `~parseme6/UD_Data/`:

cs	Czech
de	German
en	English
es	Spanish
fi	Finnish
fr	French
ga	Irish
hu	Hungarian
it	Italian

sv	Swedish

Choose a language that you feel comfortable working with and copy the corresponding training and development test files into the MaltParser working directory. For example, to work on English, run the following commands in your `maltparser-1.8.1` directory:

```
cp ~parseme6/UD_Data/en/en-ud-train.conllu .
cp ~parseme6/UD_Data/en/en-ud-dev.conllu .
```

To make sure that you are all set to start working, run the `ls` command in the `maltparser-1.8.1` directory to verify that your data files have been properly copied. If you don't see them, notify the lab instructor.

NB: From now on, we will assume that we are working on English. If you have chosen another language, you need to substitute the corresponding file names in the commands below.

3. Train a parser

Training a parser is achieved by running a command like the following:

```
java -jar -Xmx2g maltparser-1.8.1.jar -c en-parser -m learn -i en-ud-train.conllu
```

This command can be broken down into two parts, where the first is:

```
java -jar -Xmx2g maltparser-1.8.1.jar
```

This tells the Java Virtual Machine on your computer to run MaltParser with a heapspace of 2GB. (The heapspace is set by the flag `-Xmx2g`. If you forget this flag, the process will probably run out of memory.)

The rest of the command consists of specific MaltParser parameters:

`-c`	The name of the parsing model you are creating, which can be anything you like. If you name your model `en-parser`, it will be saved in a file could `en-parser.mco`
`-m`	The processing mode. This should be `learn` when you are training a model.
`-i`	The input file. This should be the name of your training set (`en-ud-train.conllu` in our example).

Training a model should take somewhere between 10 seconds and a few minutes, depending on the size of the training set. If it takes longer (or crashes), notify the lab instructor.

NB: MaltParser has a large number of options that can be used to tune the accuracy of the parser for different languages and representations. For more information, see the user guide. A quick guide to MaltParser optimization can be found here. Alternatively, you may want to use MaltOptimizer, which is an automatic optimization tool.

4. Parse some new data

Parsing a new set of sentences using the trained parser is achieved with a command like the following:

```
java -jar -Xmx2g maltparser-1.8.1.jar -c en-parser -m parse -i en-ud-dev.conllu -o en-ud-dev-parsed.conllu
```

The first part of the command is the same as for training, telling the JVM to run MaltParser, but the MaltParser parameters are slightly different:

`-c`	The name of the parsing model, which has to be trained beforehand. For example, if we specify `en-parser`, the model saved as `en-parser.mco` will be used.
`-m`	The processing mode. This should be `parse` when you are parsing new data.
`-i`	The input file. This should be the name of the input file to be parsed (`en-ud-dev.conllu` in our example).
`-o`	The output file. This should be the name of the output file where the parsers will be saved (`en-ud-dev-parsed.conllu` in our example).

5. Evaluate the parser

To evaluate the parse results, you first need to download an evaluation tool. We are going to use the MaltEval tool, which you can copy from the `~parseme6` directory:

```
cp ~parseme6/MaltEval.jar .
```

Make sure that the file `MaltEval.jar` is now in the same directory as your treebank and parsed files. You can now evaluate your parser by running:

```
java -jar -Xmx2g MaltEval.jar -g en-ud-dev.conllu -s en-ud-dev-parsed.conllu
```

The output given is the labeled attachment score (LAS) and the number of tokens included in the evaluation. There are a number of flags that can be used to get more output from MaltEval. For example, to get both LAS and UAS, add the flag:

```
--Metric 'LAS;UAS'
```

To get precision and recall for different dependency relations, add the flag:

```
--GroupBy Deprel:all
```

For more information about the options available, see the MaltEval user guide. If you want to download MaltEval later for your own uses, it is available under **Tools** on the MaltParser website maltparser.org.

6. Do an error analysis

MaltEval can also be used to visualize dependency trees with color coding of parse errors. To visualize the parse trees produced earlier, run:

```
java -jar -Xmx2g MaltEval.jar -g en-ud-dev.conllu -s en-ud-dev-parsed.conllu -v 1
```

You can use the buttons next to the list of the sentences at the bottom to navigate either to the next or previous sentence or to the next or previous parse error. You can also use the search tool to search for different linguistic phenomena specified by particular part-of-speech tags or dependency labels. For example, to search for sentences containing an instance of the `mwe` relation, select `Gold-standard` from the menu `Search in:`, select `Deprel` from the menu `Search by:`, select `mwe` from the menu `Search for:`, and push the `Search` button.

Try to find sentences containing multiword expressions and see if you can find any patterns in the parser's performance. In addition to `mwe`, which is used only for fixed expressions, you can search for `compound` (possibly with subtypes) and `name`. You can also just browse sentences and try to identify multiword expressions that are not annotated as such. Have fun!