Moses Installation and Training Run-Through

The purpose of this guide is to offer a step-by-step example of downloading, compiling, and runing the Moses decoder and related support tools. I make no claims that all of the steps here will work perfectly on every machine you try it on, or that things will stay the same as the software changes. Please remember that Moses is research software under active development.

PART I - Download and Configure Tools and Data

Support Tools Background

Moses has a number of scripts designed to aid training, and they rely on `GIZA++` and `mkcls` to function. More information on the origins of these tools is available at:

- http://www.fjoch.com/GIZA++.html
- http://www.fjoch.com/mkcls.html

A Google Code project has been set up, and the code is being maintained:

- http://giza-pp.googlecode.com/

Moses uses SRILM-style language models. SRILM is available from:

- http://www.speech.sri.com/projects/srilm/download.html

(Optional) The IRSTLM tools provide the ability to use quantized and disk memory-mapped language models. It's optional, but we'll be using it in this tutorial:

- http://sourceforge.net/projects/irstlm

Support Tools Installation

Before we start building and using the Moses codebase, we have to download and compile all of these tools. See the list of versions to double-check that you are using the same code.

I'll be working under `/home/jschroe1/demo` in these examples. I assume you've set up some appropriately named directory in your own system. I'm installing these tools under an FC6 distro.

Changes to run the same setup under Mac OS X 10.5 are highlighted. For the Mac I'm running under `/Users/josh/demo`.

Machine Translation Marathon changes are highlighted. We probably won't have time to train a full model today.

```
mkdir tools
cd tools
```

- Download and compile `GIZA++` and `mkcls`

```
•
•   wget http://giza-pp.googlecode.com/files/giza-pp-
v1.0.2.tar.gz
•   curl -O http://giza-pp.googlecode.com/files/giza-pp-
v1.0.2.tar.gz
•   tar -xzvf giza-pp-v1.0.2.tar.gz
•   cd giza-pp
```

OS X doesn't support static linking (here's more info), so we need to tweak two Makefiles. GIZA++-v2/Makefile:

```
15c15
< LDFLAGS = -static
---
> LDFLAGS =
```

mkcls-v2/Makefile is OK

```
make
```

- Copy compiled executables to `bin/` folder

```
•
•   cd ../
•   mkdir bin
•   cp giza-pp/GIZA++-v2/GIZA++ bin/
•   cp giza-pp/mkcls-v2/mkcls bin/
•   cp giza-pp/GIZA++-v2/snt2cooc.out bin/
```

- Download and compile SRILM

  SRILM has a lot of dependencies. These instructions work on `bash`.

```
mkdir srilm
cd srilm
```

(get srilm download 1.5.7, requires web registration, you'll end up with a .tgz file to copy to this directory)

```
tar -xzvf srilm.tgz
```

(SRILM expands in the current directory, not in a sub-directory).

READ THE INSTALL FILE - there are a lot of tips in there.

```
chmod +w Makefile
```

edit Makefile to point to your directory. Here's my diff:

```
7c7
< # SRILM = /home/speech/stolcke/project/srilm/devel
---
> SRILM = /home/jschroe1/demo/tools/srilm
```

```
7c7
< # SRILM = /home/speech/stolcke/project/srilm/devel
---
> SRILM = /Users/josh/demo/tools/srilm
```

```
make World
```

If you want to test that this worked, you'll need to add SRILM to your path and run their test suite. You don't need these in your path for normal training and decoding with Moses.

```
export
PATH=/home/jschroe1/demo/tools/srilm/bin/i686:/home/jschroe1/de
mo/tools/srilm/bin:$PATH
export
PATH=/Users/josh/demo/tools/srilm/bin/macosx:/Users/josh/demo/t
ools/srilm/bin:$PATH
cd test
```

OSX doesn't have `gawk`, but it does have `awk`. Change the following:

```
chmod +w go.run-test

19c19
< diff="gawk -f compare-outputs 2>/dev/null"
---
> diff="awk -f compare-outputs 2>/dev/null"
```

```
make all
```

Check output, look for `IDENTICAL` and `DIFFERS`. I still see the occasional difference, but it's pretty easy to tell when the tools are working and when they're dying instantly.

- Download and compile IRSTLM

  You can either download a release or check out the latest files from svn.

```
cd /home/jschroe1/demo/tools
wget http://downloads.sourceforge.net/irstlm/irstlm-5.20.00.tgz
curl -LO http://downloads.sourceforge.net/irstlm/irstlm-
5.20.00.tgz
tar -xzvf irstlm-5.20.00.tgz
```

  Or get it from sourceforge:

```
mkdir irstlm
svn co https://irstlm.svn.sourceforge.net/svnroot/irstlm irstlm

cd irstlm
./install
OSTYPE=darwin ./install
```

  On my system, Moses looks in `irstlm/bin/i686`, and IRST compiles to `irstlm/bin/i686-redhat-linux-gnu`. Symlink to fix.

```
cd bin
ln -s i686-redhat-linux-gnu i686
cd ../../
```

Get The Latest Moses Version

Moses is available via Subversion from Sourceforge. See the list of versions to double-check that you are using the same code as this example. From the `tools/` directory:

```
mkdir moses
svn co
https://mosesdecoder.svn.sourceforge.net/svnroot/mosesdecoder/trunk
moses
```

This will copy all of the Moses source code to your local machine.

Compile Moses

Within the Moses folder structure are projects for Eclipse, Xcode, and Visual Studio -- though these are not well maintained and may not be up to date. I'll focus on the linux command-line method, which is the preferred way to compile.

For OS X versions 10.4 and lower, you need to upgrade `aclocal` and `automake` to at least version 1.9 (1.6 is the default in 10.4) and set the variables `ACLOCAL` and `AUTOMAKE` in `./regenerate-makefiles.sh`.

```
cd moses
./regenerate-makefiles.sh
./configure --with-srilm=/home/jschroe1/demo/tools/srilm --with-
irstlm=/home/jschroe1/demo/tools/irstlm
make -j 2
```

(The `-j 2` is optional. `make -j X` where X is number of simultaneous tasks is a speedier option for machines with multiple processors)

This creates several files we will be using:

- `misc/processPhraseTable` - Used to binarize phrase tables
- `misc/processLexicalTable` - Used to binarize reordering tables
- `moses-cmd/src/moses` - The actual decoder

Confirm Setup Success

A sample model capable of translating one sentence is available on the Moses website. Download it and translate the sample input file.

```
cd /home/jschroe1/demo/
mkdir data
cd data
wget http://www.statmt.org/moses/download/sample-models.tgz
curl -O http://www.statmt.org/moses/download/sample-models.tgz
tar -xzvf sample-models.tgz
cd sample-models/phrase-model/
../../../tools/moses/moses-cmd/src/moses -f moses.ini < in > out
```

The input has "das ist ein kleines haus" listed twice, so the output file (`out`) should contain "this is a small house" twice.

At this point, it might be wise for you to experiment with the command line options of the Moses decoder. A tutoral using this example model is available at http://www.statmt.org/moses/?n=Moses.Tutorial.

Compile Moses Support Scripts

Moses uses a set of scripts to support training, tuning, and other tasks. The support scripts used by Moses are "released" by a Makefile which edits their paths to match your local environment. First, make a place for the scripts to live:

```
cd ../../../tools/
mkdir moses-scripts
cd moses/scripts
```

edit Makefile as needed. Here's my diff:

```
13,14c13,14
< TARGETDIR?=/home/s0565741/terabyte/bin
< BINDIR?=/home/s0565741/terabyte/bin
---
> TARGETDIR?=/home/jschroe1/demo/tools/moses-scripts
> BINDIR?=/home/jschroe1/demo/tools/bin

make release
```

This will create a time-stamped folder named `/home/jschroe1/demo/moses-scripts/scripts-`*`YYYYMMDD-HHMM`* with released versions of all the scripts. You will call these versions when training and tuning Moses. Some Moses training scripts also require a `SCRIPTS_ROOTDIR` environment variable to be set. The output of `make release` should indicate this. Most scripts allow you to override this by setting a `-scripts-root-dir` flag or something similar.

```
export SCRIPTS_ROOTDIR=/home/username/lab4/moses-scripts/scripts-
YYYYMMDD-HHMM
```

Additional Scripts

There are few scripts not included with moses which are useful for preparing data. These were originally made available as part of the WMT08 Shared Task and Europarl v3 releases, I've consolidated some of them into one set.

```
cd ../../
wget http://homepages.inf.ed.ac.uk/jschroe1/how-to/scripts.tgz
curl -O http://homepages.inf.ed.ac.uk/jschroe1/how-to/scripts.tgz
tar -xzvf scripts.tgz
```

We'll also get a NIST scoring tool.

```
wget ftp://jaguar.ncsl.nist.gov/mt/resources/mteval-v11b.pl
```

On the Mac, use `ftp` or a web browser to get the file. `curl` and I had a fight about it.

```
chmod +x mteval-v11b.pl
```

PART II - Build a Model

We'll used the WMT08 News Commentary data set, about 55k sentences. This should be good enough for moderate quality but still be doable in a reasonable amount of time on most machines. For this example we'll use FR-EN.

```
cd ../data
wget http://www.statmt.org/wmt08/training-parallel.tar
curl -O http://www.statmt.org/wmt08/training-parallel.tar
tar -xvf training-parallel.tar --wildcards training/news-
commentary08.fr-en.*
```

If you're low on disk space, remove the full tar.

```
rm training-parallel.tar

cd ../
```

Prepare Data

First we'll set up a working directory where we'll store all the data we prepare.

```
mkdir work
```

- **Tokenize training data**

  We'll keep the initial versions in zipped format. Note that Mac uses `gzcat` instead of `zcat`, so we'll just use `gzip -cd` for both.

  ```
  mkdir work/corpus
  gzip -cd data/training/news-commentary08.fr-en.fr.gz |
  tools/scripts/tokenizer.perl -l fr > work/corpus/news-
  commentary.tok.fr
  gzip -cd data/training/news-commentary08.fr-en.en.gz |
  tools/scripts/tokenizer.perl -l en > work/corpus/news-
  commentary.tok.en
  ```

- **Filter out long sentences**

  ```
  •
  •   tools/moses-scripts/scripts-YYYYMMDD-HHMM/training/clean-
  corpus-n.perl work/corpus/news-commentary.tok fr en
  work/corpus/news-commentary.clean 1 40
  ```

  This ensures that only sentences of length 1-40 are selected for training. In this case, we lose almost 11,000 sentences:

```
Input sentences: 55030  Output sentences:  44219
```

We do this because GIZA++ takes a very long time to train on long sentences. This isn't much of an issue with a 55,000-sentence corpus, but it can be a limitation when dealing with corpora of millions of sentences. Of course, the more data you throw out to improve training times, the less examples Moses can choose from when building translations.

- **Lowercase training data**

```
•
•   tools/scripts/lowercase.perl < work/corpus/news-
commentary.clean.fr > work/corpus/news-commentary.lowercased.fr
•   tools/scripts/lowercase.perl < work/corpus/news-
commentary.clean.en > work/corpus/news-commentary.lowercased.en
```

Build Language Model

Language models are concerned only with n-grams in the data, so sentence length doesn't impact training times as it does in GIZA++. So, we'll lowercase the full 55,030 tokenized sentences to use for language modeling. Many people incorporate extra target language monolingual data into their language models.

```
mkdir work/lm
tools/scripts/lowercase.perl < work/corpus/news-commentary.tok.en >
work/lm/news-commentary.lowercased.en
```

We will use SRILM to build a tri-gram language model.

```
tools/srilm/bin/i686/ngram-count -order 3 -interpolate -kndiscount -
unk -text work/lm/news-commentary.lowercased.en -lm work/lm/news-
commentary.lm
tools/srilm/bin/macosx/ngram-count -order 3 -interpolate -kndiscount
-unk -text work/lm/news-commentary.lowercased.en -lm work/lm/news-
commentary.lm
```

We can see how many n-grams were created

```
head -n 5 work/lm/news-commentary.lm


\data\
ngram 1=36035
ngram 2=411595
ngram 3=118368
```

Train Phrase Model

Moses' toolkit does a great job of wrapping up calls to `mkcls` and `GIZA++` inside a training script, and outputting the phrase and reordering tables needed for decoding. The script that does this is called train-factored-phrase-model.perl

> If you want to skip this step, you can use the pre-prepared model and ini files located at `/afs/ms/u/m/mtm52/BIG/work/model/moses.ini` and `/afs/ms/u/m/mtm52/BIG/work/model/moses-bin.ini` instead of the local references used in this tutorial. Move on to sanity checking your setup.

We'll run this in the background and nice it since it'll peg the CPU while it runs. It may take up to an hour, so this might be a good time to run through the tutorial page mentioned earlier using the `sample-models` data.

```
nohup nice tools/moses-scripts/scripts-YYYYMMDD-HHMM/training/train-
factored-phrase-model.perl -scripts-root-dir tools/moses-
scripts/scripts-YYYYMMDD-HHMM/ -root-dir work -corpus
work/corpus/news-commentary.lowercased -f fr -e en -alignment grow-
diag-final-and -reordering msd-bidirectional-fe -lm
0:3:/home/jschroe1/demo/work/lm/news-commentary.lm >&
work/training.out &
nohup nice tools/moses-scripts/scripts-YYYYMMDD-HHMM/training/train-
factored-phrase-model.perl -scripts-root-dir tools/moses-
scripts/scripts-YYYYMMDD-HHMM/ -root-dir work -corpus
work/corpus/news-commentary.lowercased -f fr -e en -alignment grow-
diag-final-and -reordering msd-bidirectional-fe -lm
0:3:/Users/josh/demo/work/lm/news-commentary.lm >& work/training.out
&
```

You can `tail -f work/training.out` file to watch the progress of the tuning script. The last step will say something like:

```
(9) create moses.ini @ Tue Jan 27 19:40:46 CET 2009
```

Now would be a good time to look at what we've done.

```
cd work
ls
corpus  giza.en-fr  giza.fr-en  lm  model
```

We'll look in the model directory. The three files we really care about are in bold.

```
cd model
ls -l
total 192554
-rw-r--r-- 1 jschroe1 people  5021309 Jan 27 19:23 aligned.grow-diag-
final-and
-rw-r--r-- 1 jschroe1 people 27310991 Jan 27 19:24 extract.gz
```

```
-rw-r--r-- 1 jschroe1 people 27043024 Jan 27 19:25 extract.inv.gz
-rw-r--r-- 1 jschroe1 people 21069284 Jan 27 19:25 extract.o.gz
-rw-r--r-- 1 jschroe1 people  6061767 Jan 27 19:23 lex.e2f
-rw-r--r-- 1 jschroe1 people  6061767 Jan 27 19:23 lex.f2e
-rw-r--r-- 1 jschroe1 people     1032 Jan 27 19:40 moses.ini
-rw-r--r-- 1 jschroe1 people 67333222 Jan 27 19:40 phrase-table.gz
-rw-r--r-- 1 jschroe1 people 26144298 Jan 27 19:40 reordering-
table.gz
```

Memory-Map LM and Phrase Table (Optional)

The language model and phrase table can be memory-mapped on disk to minimize the amount of RAM they consume. This isn't really necessary for this size of model, but we'll do it just for the experience.

More information is available on the Moses' web site at: http://www.statmt.org/moses/?n=Moses.AdvancedFeatures and http://www.statmt.org/moses/?n=FactoredTraining.BuildingLanguageModel .

Performing these steps can lead to heavy disk use during decoding - you're basically using your hard drive as RAM. Proceed at your own risk, especially if you're using a (slow) networked drive.

- **IRSTLM Binary Language Model**

  Produces a compact file on disk

```
cd ../../
tools/irstlm/bin/i686/compile-lm work/lm/news-commentary.lm
work/lm/news-commentary.blm
tools/irstlm/bin/i386-apple-darwin9.0/compile-lm work/lm/news-
commentary.lm work/lm/news-commentary.blm
```

- **IRSTLM Memory Mapping**

  Changing the suffix of this file to `.mm` forces the decoder to leave the file on disk instead of loading it into memory. We'll just make a symlink.

```
cd work/lm
ln -s news-commentary.blm news-commentary.blm.mm
cd ../../
```

  A note on memory mapping: IRSTLM makes use of a temp directory during decoding. Version 5.20.00 has this hard-coded to `/tmp`, but the trunk on svn has been updated to allow you to set it using the `TMP` environment variable. If this is important to your setup, be sure to set this variable, or check that it is already set appropriately.

- **Binary Phrase Table**

  As with the LM, the phrase table can be processed and read from disk on-demand instead of being loaded in its entirety into memory.

  Note that if your phrase table was not sorted, you would need to pipe the zcat through a `sort`, and use the `LC_ALL=C` flag. Depending on the size of your temp directory, you may have to have sort use a different directory using the `-T` flag. `man sort` for more info.

```
gzip -cd work/model/phrase-table.gz | LC_ALL=C sort |
tools/moses/misc/processPhraseTable -ttable 0 0 - -nscores 5 -
out work/model/phrase-table
```

- **Binary Reordering Table**

  Similar to the phrase table, including optional sorting.

```
gzip -cd work/model/reordering-table.gz | LC_ALL=C sort |
tools/moses/misc/processLexicalTable -out
work/model/reordering-table
```

- **Edit Config File**

  We'll make a copy of `work/model/moses.ini` and set it to use these files. Moses will automatically use binary phrase and reordering tables if they are present with the correct naming stem, and since we used the same stem for output as for our input tables, we just need to remove the `.gz` suffix. For LM information, we need to set the type to be IRSTLM (1) instead of SRILM (0) and change the LM file.

```
cp work/model/moses.ini work/model/moses-bin.ini
```

  Here's my diff:

```
15c15
< 0 0 5 /home/jschroe1/demo/work/model/phrase-table.gz
---
> 0 0 5 /home/jschroe1/demo/work/model/phrase-table
21c21
< 0 0 3 /home/jschroe1/demo/work/lm/news-commentary.lm
---
> 1 0 3 /home/jschroe1/demo/work/lm/news-commentary.blm.mm
31c31
< 0-0 msd-bidirectional-fe 6
/home/jschroe1/demo/work/model/reordering-table.gz
---
```

```
> 0-0 msd-bidirectional-fe 6
/home/jschroe1/demo/work/model/reordering-table
```

Sanity Check Trained Model

We haven't tuned yet, but let's just check that the decoder works, and output a lot of logging data with `-v 2`.

Here's an excerpt of moses initializing with binary files in place (note bold lines, and recall the IRSTLM `TMP` issue):

```
echo "c' est une petite maison ." | TMP=/tmp tools/moses/moses-
cmd/src/moses -f work/model/moses-bin.ini
Loading lexical distortion models...
have 1 models
Creating lexical reordering...
weights: 0.300 0.300 0.300 0.300 0.300 0.300
binary file loaded, default OFF_T: -1
Created lexical orientation reordering
Start loading LanguageModel /home/jschroe1/demo/work/lm/news-
commentary.blm.mm : [0.000] seconds
In LanguageModelIRST::Load: nGramOrder = 3
Loading LM file (no MAP)
blmt
loadbin()
mapping 36035 1-grams
mapping 411595 2-grams
mapping 118368 3-grams
done
OOV code is 1468
IRST: m_unknownId=1468
Finished loading LanguageModels : [0.000] seconds
Start loading PhraseTable
/amd/nethome/jschroe1/demo/work/model/phrase-table.0-0 : [0.000]
seconds
using binary phrase tables for idx 0
reading bin ttable
size of OFF_T 8
binary phrasefile loaded, default OFF_T: -1
Finished loading phrase tables : [1.000] seconds
IO from STDOUT/STDIN
```

And here's one if you skipped the memory mapping steps:

```
echo "c' est une petite maison ." | tools/moses/moses-cmd/src/moses -
f work/model/moses.ini
Loading lexical distortion models...
have 1 models
Creating lexical reordering...
weights: 0.300 0.300 0.300 0.300 0.300 0.300
Loading table into memory...done.
Created lexical orientation reordering
Start loading LanguageModel /home/jschroe1/demo/work/lm/news-
commentary.lm : [47.000] seconds
```

```
/home/jschroe1/demo/work/lm/news-commentary.lm: line 1476: warning:
non-zero probability for <unk> in closed-vocabulary LM
Finished loading LanguageModels : [49.000] seconds
Start loading PhraseTable
/amd/nethome/jschroe1/demo/work/model/phrase-table.0-0.gz : [49.000]
seconds
Finished loading phrase tables : [259.000] seconds
IO from STDOUT/STDIN
```

Again, while these short load times and small memory footprint are nice, decoding times will be slower with memory-mapped models due to disk access.

PART III - Prepare Tuning and Test Sets

Prepare Data

We'll use some of the dev and devtest data from WMT08. We'll stick with news-commentary data and use dev2007 and test2007. We only need to look at the input (FR) side of our testing data.

- **Download tuning and test sets**

```
•
•   cd data/
•   wget http://www.statmt.org/wmt08/devsets.tgz
•   curl -O http://www.statmt.org/wmt08/devsets.tgz
•   tar -xzvf devsets.tgz
•   cd ../
```

- **Tokenize sets**

```
•
•   mkdir work/tuning
•   tools/scripts/tokenizer.perl -l fr < data/dev/nc-dev2007.fr >
  work/tuning/nc-dev2007.tok.fr
•   tools/scripts/tokenizer.perl -l en < data/dev/nc-dev2007.en >
  work/tuning/nc-dev2007.tok.en
•   mkdir work/evaluation
•   tools/scripts/tokenizer.perl -l fr < data/devtest/nc-
  test2007.fr > work/evaluation/nc-test2007.tok.fr
```

- **Lowercase sets**

```
•
•   tools/scripts/lowercase.perl < work/tuning/nc-dev2007.tok.fr
  > work/tuning/nc-dev2007.lowercased.fr
•   tools/scripts/lowercase.perl < work/tuning/nc-dev2007.tok.en
  > work/tuning/nc-dev2007.lowercased.en
•   tools/scripts/lowercase.perl < work/evaluation/nc-
  test2007.tok.fr > work/evaluation/nc-test2007.lowercased.fr
```

PART IV - Tuning

Note that this step can take many hours, even days, to run on large phrase tables and tuning sets. We'll use the non-memory-mapped versions for decoding speed. The training script controls for large phrase and reordering tables by filtering them to include only data relevant to the tuning set (we'll do this ourselves for the test data later).

```
nohup nice tools/moses-scripts/scripts-YYYYMMDD-HHMM/training/mert-
moses.pl work/tuning/nc-dev2007.lowercased.fr work/tuning/nc-
dev2007.lowercased.en tools/moses/moses-cmd/src/moses
work/model/moses.ini --working-dir work/tuning/mert --rootdir
/home/jschroe1/demo/tools/moses-scripts/scripts-YYYYMMDD-HHMM/ --
decoder-flags "-v 0" >& work/tuning/mert.out &
```

Since this can take so long, we can instead make a small, 100 sentence tuning set just to see if the tuning process works. This won't generate very good weights, but it will let us confirm that our tools work.

```
head -n 100 work/tuning/nc-dev2007.lowercased.fr > work/tuning/nc-
dev2007.lowercased.100.fr
head -n 100 work/tuning/nc-dev2007.lowercased.en > work/tuning/nc-
dev2007.lowercased.100.en
nohup nice tools/moses-scripts/scripts-YYYYMMDD-HHMM/training/mert-
moses.pl work/tuning/nc-dev2007.lowercased.100.fr work/tuning/nc-
dev2007.lowercased.100.en tools/moses/moses-cmd/src/moses
work/model/moses.ini --working-dir work/tuning/mert --rootdir
/home/jschroe1/demo/tools/moses-scripts/scripts-YYYYMMDD-HHMM/ --
decoder-flags "-v 0" >& work/tuning/mert.out &
```

(Note that the scripts rootdir path needs to be absolute).

While this runs, check out the contents of `work/tuning/mert`. You'll see a set of runs, n-best lists for each, and `run*.moses.ini` files showing the weights used for each file. You can see the score each run is getting by looking at the last line of each `run*.cmert.log` file

```
cd work/tuning/mert
tail -n 1 run*.cmert.log

==> run1.cmert.log <==
Best point: 0.028996 0.035146 -0.661477 -0.051250 0.001667 0.056762
0.009458 0.005504 -0.006458 0.029992 0.009502 0.012555 0.000000 -
0.091232 => 0.282865

==> run2.cmert.log <==
Best point: 0.056874 0.039994 0.046105 -0.075984 0.032895 0.020815 -
0.412496 0.018823 -0.019820 0.038267 0.046375 0.011876 -0.012047 -
0.167628 => 0.281207
```

```
==> run3.cmert.log <==
Best point: 0.041904 0.030602 -0.252096 -0.071206 0.012997 0.516962
0.001084 0.010466 0.001683 0.008451 0.001386 0.007512 -0.014841 -
0.028811 => 0.280953

==> run4.cmert.log <==
Best point: 0.088423 0.118561 0.073049 0.060186 0.043942 0.293692 -
0.147511 0.037605 0.008851 0.019371 0.015986 0.018539 0.001918 -
0.072367 => 0.280063

==> run5.cmert.log <==
Best point: 0.059100 0.049655 0.187688 0.010163 0.054140 0.077241
0.000584 0.101203 0.014712 0.144193 0.219264 -0.005517 -0.047385 -
0.029156 => 0.280930
```

This gives you an idea if the system is improving or not. You can see that in this case it isn't, because we don't have enough data in our system and we haven't let tuning run for enough iterations. Kill `mert-moses.pl` after a few iterations just to get some weights to use.

If mert were to finish successfully, it would create a file named `work/tuning/mert/moses.ini` containing all the weights we needed. Since we killed mert, copy the best moses.ini config to be the one we'll use. Note that the weights calculated in `run1.cmert.log` were used to make the config file for run2, so we want `run2.moses.ini`

If you want to use the weights from a finished mert run, try `/afs/ms/u/m/mtm52/BIG/work/tuning/mert/moses.ini`

```
cp run2.moses.ini moses.ini
```

Insert weights into configuration file

```
cd ../../../
tools/scripts/reuse-weights.perl work/tuning/mert/moses.ini <
work/model/moses.ini > work/tuning/moses-tuned.ini
tools/scripts/reuse-weights.perl work/tuning/mert/moses.ini <
work/model/moses-bin.ini > work/tuning/moses-tuned-bin.ini
```

PART V - Filtering Test Data

Filtering is another way, like binarizing, to help reduce memory requirements. It makes smaller phrase and reordering tables that contain only entries that will be used for a particular test set. Binarized models don't need to be filtered since they don't take up RAM when used. Moses has a script that does this for us, which we'll apply to the evaluation test set we prepared earlier:

```
tools/moses-scripts/scripts-YYYYMMDD-HHMM/training/filter-model-
given-input.pl  work/evaluation/filtered.nc-test2007
work/tuning/moses-tuned.ini work/evaluation/nc-test2007.lowercased.fr
```

There is also a `filter-and-binarize-model-given-input.pl` script if your filtered table would still be too large to load into memory.

PART VI - Run Tuned Decoder on Development Test Set

We'll try this a few ways.

- First, reusing the weights from tuning, without filtering:

> I'd skip this step today. It takes too much RAM on the lab machines.

```
nohup nice tools/moses/moses-cmd/src/moses -config
work/tuning/moses-tuned.ini -input-file work/evaluation/nc-
test2007.lowercased.fr 1> work/evaluation/nc-
test2007.tuned.output 2> work/evaluation/tuned.decode.out &
```

- Next, with the filtered phrase table from the output of the filtering step:

```
•
•   nohup nice tools/moses/moses-cmd/src/moses -config
work/evaluation/filtered.nc-test2007/moses.ini -input-file
work/evaluation/nc-test2007.lowercased.fr 1>
work/evaluation/nc-test2007.tuned-filtered.output 2>
work/evaluation/tuned-filtered.decode.out &
```

- Finally, if you performed binarizing, you can try that too:

```
•
•   TMP=/tmp nohup nice tools/moses/moses-cmd/src/moses -config
work/tuning/moses-tuned-bin.ini -input-file work/evaluation/nc-
test2007.lowercased.fr 1> work/evaluation/nc-test2007.tuned-
bin.output 2> work/evaluation/tuned-bin.decode.out &
```

All three of these outputs should be identical, but they will take different amounts of time and memory to compute.

> If you don't have time to run a full decoding session, you can use an output located at `/afs/ms/u/m/mtm52/BIG/work/evaluation/nc-test2007.tuned-filtered.output`

PART VII - Evaluation

Train Recaser

Now we'll train a recaser. It uses a statistical model to "translate" between lowercased and cased data.

```
mkdir work/recaser
tools/moses-scripts/scripts-YYYYMMDD-HHMM/recaser/train-recaser.perl
-train-script tools/moses-scripts/scripts-YYYYMMDD-
HHMM/training/train-factored-phrase-model.perl -ngram-count
tools/srilm/bin/i686/ngram-count -corpus work/corpus/news-
commentary.tok.en -dir /home/jschroe1/demo/work/recaser -scripts-
root-dir tools/moses-scripts/scripts-YYYYMMDD-HHMM/
```

This goes through a whole GIZA and LM training run to go from lowercase sentences to cased sentences. Note that the `-dir` flag needs to be absolute.

Recase the output

```
tools/moses-scripts/scripts-YYYYMMDD-HHMM/recaser/recase.perl -model
work/recaser/moses.ini -in work/evaluation/nc-test2007.tuned-
filtered.output -moses tools/moses/moses-cmd/src/moses >
work/evaluation/nc-test2007.tuned-filtered.output.recased
```

Detokenize the output

```
tools/scripts/detokenizer.perl -l en < work/evaluation/nc-
test2007.tuned-filtered.output.recased > work/evaluation/nc-
test2007.tuned-filtered.output.detokenized
```

Wrap the output in XML

```
tools/scripts/wrap-xml.perl data/devtest/nc-test2007-ref.en.sgm en
my-system-name < work/evaluation/nc-test2007.tuned-
filtered.output.detokenized > work/evaluation/nc-test2007.tuned-
filtered.output.sgm
```

Score with NIST-BLEU

```
tools/mteval-v11b.pl -s data/devtest/nc-test2007-src.fr.sgm -r
data/devtest/nc-test2007-ref.en.sgm -t work/evaluation/nc-
test2007.tuned-filtered.output.sgm -c

  Evaluation of any-to-en translation using:
    src set "nc-test2007" (1 docs, 2007 segs)
    ref set "nc-test2007" (1 refs)
    tst set "nc-test2007" (1 systems)
```

```
NIST score = 6.9126  BLEU score = 0.2436 for system "my-system-name"
```

We got a BLEU score of 24.4! Hooray! Best translations ever! Let's all go to the pub!

Appendix A - Versions

- GIZA++ and mkcls: Google Code 1.0.2
- SRILM: 1.5.7
- IRSTLM: 5.20.00, or -r 232 from svn
- Moses: -r 2014 from svn