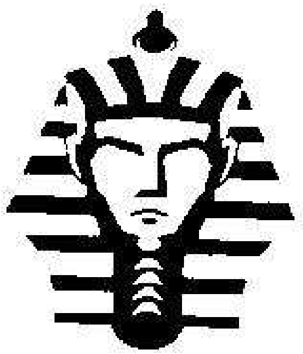
PHARAOH



a Beam Search Decoder for Phrase-Based Statistical Machine Translation Models

User Manual and Description

Philipp Koehn
koehn@isi.edu
USC Information Sciences Institute

December 11, 2003

Abstract

This document describes Pharaoh, a beam search decoder for phrase-based statistical machine translation models. It is both a user manual that describes how to run the program to perform machine translation, as well as description of the mechanics of the decoder.

We developed Pharaoh as part of our PhD thesis [Koehn, 2003] at the University of Southern California. It is an implementation of the currently best-performing methods for statistical machine translation: heuristic beam search and phrase-based translation. Experiments with this decoder were also reported by Koehn et al. [2003].

The decoder allows you to supply your own translation model in simple format and translate text between languages. For full use of the decoder you will also need the SRI language modeling toolkit and (for n-best list generating) the Carmel finite state machine toolkit. Both of these resources are available for free. We do not include a program to learn a translation table from a parallel corpus, but we do describe methods how to do this in this document.

Contents

1 User Manual

This document consists of three parts, (1) a user manual for the decoder, (2) an introduction to phrase-based statistical machine translation, and (3) a description of the decoder. Each of these three sections is self-contained and highlights a different aspect of the same subject matter.

We start in this section with a manual on using the decoder, since we expect this to be closest to the interests of the reader. While we try to make this as self-contained as possible, some details will be hard to understand without knowledge of the other sections. If you have more time, or if you are more interested in the theoretical background, it might be better to start with Section 2 and 3.

1.1 Download

The decoder code is available at the web pages of the Information Sciences Institute at the University of Southern Calfornia:

`http://www.isi.edu/licensed-sw/pharaoh/`

After unpacking, you will see the following directories and files:

`bin/pharaoh` the binary of the decoder (Linux/Intel)
`model` a toy example that is used in this manual
`doc` a directory that includes this documentation
`lm/europarl.srilm` a language model for use with the models

The package contains the decoder code, along with documentation and a toy translation model. In the following we will walk through the features of the decoder. The reader is advised to download and unpack the code and repeat the steps and try variations.

What is missing to build a full machine translation system from scratch?

- A **parallel corpus** can be obtained from various sources, e.g., the Europarl corpus [Koehn, 2002] from

  `http://www.isi.edu/~koehn/europarl`

- The program to train a **language model** is available from SRI at

  `http://www.speech.sri.com/projects/srilm/`

- For the generation of word lattices and n-best list, you will need the **Carmel finite state toolkit**, which is available at

  `http://www.isi.edu/licensed-sw/carmel/`

- A **training system** to generate translation models for the decoder. This is currently not available, but this document describes methods how to build translation models, which should enable the reader to write her own code for this.

1.2 A Simple Translation Model

Let us begin with a look at the toy translation model that is provided with the decoder release. The directory `model` contains three files

`toy.f2n`	the phrase translation table $p(n|f)$
`toy.n2f`	the phrase translation table $p(f|n)$
`pharaoh.ini`	the configuration file for the decoder

Let's look at the first line of the phrase translation table $p(n|f)$ (file `toy.f2n`):

```
the ||| der ||| 1.0
```

This entry means that the probality of translating the German (or foreign, or f) word *der* into the English (or native, or n) word *the* is 1.0. Or in mathematical notation: $p(\text{the}|\text{der}) = 1.0$.

Now, let's take a look at the first line of the other phrase translation table, $p(f|n)$ (file `toy.n2f`):

```
der ||| the ||| 0.3
```

Here we have $p(\text{der}|\text{the}) = 0.3$. Clearly $p(n|f) \neq p(f|n)$. This is generally true, the reason being here, that there are multiple German definite determiners, while English has only *the*. Look at the others in the translation table file `toy.n2f`:

```
der ||| the ||| 0.3
das ||| the ||| 0.4
die ||| the ||| 0.3
```

The translation tables are the main knowledge source for the machine translation decoder. The decoder consults these tables to figure out how to translate input in one language (the foreign language, or f) to another language (the native language, or n).

Being a phrase translation model, the translation tables do not only contain single word entries, but multi-word entries. These are called phrases, but this concept means nothing more than an arbitrary sequence of words, with no sophisticated linguistic motivation.

Here is an example for a phrase translation entry in `toy.f2n`:

```
this is ||| das ist ||| 0.3
```

Why are there two phrase translation tables? Because of the source channel model, the translation model is reformulated from $p(n|f)$ to $p(n)p(f|n)$, allowing for the use of the language model $p(n)$. In practice that means that the "normal" translation table $p(f|n)$ is used by the decoder to find the n-best translations for an input phrase n, while the "inverse" translation table $p(n|f)$ is used for scoring. This will become more clear after reading the background on the phrase model.

1.3 Running the Decoder

Without further ado, let us run the decoder:

```
# Sample configuration file

[ttable-file-f2n]
model/toy.f2n

[ttable-file-n2f]
model/toy.n2f

[lmodel-file]
lm/europarl.srilm

[ttable-limit]
10

[ttable-threshold]
0.001

[weight-d]
1

[weight-l]
1

[weight-t]
1

[weight-w]
0
```

Figure 1: Sample configuration file for the decoder: It specifies the file locations for translation tables language model, as well as other decoder

```
% echo 'das ist ein kleines haus' | bin/pharaoh -f model/pharaoh.ini > out
Pharaoh v1.0pre7, written by Philipp Koehn
a beam search decoder for phrase-based statistical machine translation models
(c) 2002-2003 University of Southern California
loading phrase translation table from model/toy.f2n
loading inverse phrase translation table from model/toy.n2f
loading language model from lm/europarl.srilm
reading input sentences
translating.

% cat out
this is a small house
```

Here, the toy model managed to translate the German input sentence *das ist ein kleines haus* into the English *this is a small house*, which is a correct translation.

The decoder is controlled by the configuration file `model/pharaoh.ini`. This file is displayed in Figure 1 on the preceding page. We will take a look at all the parameters that are specified here (and then some) below. At this point, let us just note that the translation model files and the language model file are specified here. In this example, the file names are relative paths, but usually having full paths is better, so that the decoder does not have to be run from a specific directory.

All parameters can be specfied in this file, or on the command line. For instance, we can also indicate the language model file on the command line:

```
% bin/pharaoh -f model/pharaoh.ini -lmodel-file lm/europarl.srilm
```

We just ran the decoder on a single sentence provided on the command line. Usually we want to translate more than one sentence. In this case, the input sentences are stored in a file, one sentence per line. This file is piped into the decoder and the output is piped into some output file for further processing:

```
% pharaoh -f /somewhere/pharaoh.ini < in > out
```

Parameters
`-f, -config` – specifies the location of the configuration file

1.3.1 Trace

How the decoder works is described in detail in Section 3. But let us first develop an intuition by looking under the hood. There are two switches that force the decoder to reveal more about its inner workings: `-trace` and `-verbose`.

The trace option reveals which phrase translations were used in the best translation found by the decoder. Running the decoder with the trace switch (short `-t`) on the same example

```
echo 'das ist ein kleines haus' | bin/pharaoh -f model/pharaoh.ini -t >out
```

gives us the extended output

```
this is |0.014086|0|1| a |0.188447|2|2| small |0.000706353|3|3|
   house |1.46468e-07|4|4|
```

Each generated English phrase is now annotated with additional information:

- *this is* was generated from the German words 0–1 (*das ist*), with combined model cost of 0.014086.
- *a* was generated from the German words 2–2 (*ein*), with combined model cost of 0.188447.
- *small* was generated from the German words 3–3 (*kleines*), with combined model cost of 0.000706353.
- *house* was generated from the German words 4–4 (*haus*), with combined model cost of 1.46468e-07.

Note that the German sentence does not have to be translated in sequence. Here an example, were the English output is reordered:

```
% echo 'ein haus ist das' | bin/pharaoh -f model/pharaoh.ini -t -d 0.5
this |0.00104332|3|3| is |0.13853|2|2| a |0.0420482|0|0| house |5.85786e-08|1|1|
```

Parameters
`-t`, `-trace` – output is annotated with phrase translations used

1.3.2 Verbose

Now for the next switch, `-verbose` (short `-v`), that displays additional run time information. The verbosity of the decoder output exists in three levels. The default is 1. Moving on to `-v 2` gives a screen dump of all parameter values and summary statistics for each translated sentences:

```
echo 'das ist ein kleines haus' | bin/pharaoh -f model/pharaoh.ini -v 2
...
collected 12 translation options
HYP: 557 added, 32 discarded below threshold, 199 pruned, 572 merged.
BEST: this is a small house -28.9234
```

These mysterious statistics summarize how many internal states were generated and gives us also the probability score of the best translation: exp(-28.9234). The displayed statistics mean: 12 translation options (phrase translation table entries) were used in the search for the best translation for this sentence. 557 hypotheses were added to hypothesis stacks, 32 were not added, because they fell out of the beam, 199 were discarded from the hypothesis stacks, because better hypotheses entered the stack, and 572 hypotheses were recombined. Confused? The section on the deocder (Section 3) explains what all this means.

The most verbose output (`-v 3`) provides even more information. In fact, it is so much, that we could not possibly fit it in this manual. Run the following command and enjoy:

```
echo 'das ist ein kleines haus' | bin/pharaoh -f model/pharaoh.ini -v 3
```

Let us look together at some highlights. Before decoding, the phrase translation table is consulted for possible phrase translations. For some phrases, we find entries, for others we find nothing. Here an excerpt:

```
[das;2]
   the<1>, pC=-0.916291, c=-5.78855
   it<2>, pC=-2.30259, c=-8.0761
   this<3>, pC=-2.30259, c=-8.00205
...
[das ist;6]
   it is<6>, pC=-1.60944, c=-10.207
   this is<7>, pC=-0.223144, c=-10.2906
...
[das ist ein;0]
```

The first number next to a phrase is an internal phrase identifier, the `pC` denotes the log of the phrase translation probability, after `c` the future cost estimate for the phrase is given.

Future cost is an estimate of how hard it is to translate different parts of the sentence. After looking up phrase translation probabilities, future costs are computed for all contigous spans over the sentence:

```
future costs from 0 to 0 is -5.78855
future costs from 0 to 1 is -10.207
future costs from 0 to 2 is -15.7221
future costs from 0 to 3 is -25.4433
future costs from 0 to 4 is -34.7094
future costs from 1 to 1 is -4.92223
future costs from 1 to 2 is -10.4373
future costs from 1 to 3 is -20.1585
future costs from 1 to 4 is -29.4246
future costs from 2 to 2 is -5.5151
future costs from 2 to 3 is -15.2363
future costs from 2 to 4 is -24.5023
future costs from 3 to 3 is -9.72116
future costs from 3 to 4 is -18.9872
future costs from 4 to 4 is -9.26607
```

Some parts of the sentence are easier to translate than others. For instance the estimate for translating the first two words (0–1: *das ist*) is deemed to be cheaper (-10.207) than the last two (4–5: *kleines haus*, -18.9872). Again, the negative numbers are log probabilities. For more on future cost, please see Section 3.5 on page 32.

After all this preperation, we start to create partial translations by translating a phrase at a time. The first hypothesis is generated by translating the first German word as *the*:

```
creating hypothesis 1 from 0
   base score 0
   translation cost -0.916291
   distortion cost 0
   language model cost for 'the' -2.03434
   word penalty -0
   score -2.95064 + futureCost -29.4246 = -32.3752
```

Here, starting with the empty initial hypothesis 0, a new hypothesis (id 1) is created. Starting from zero cost (base score), translating the phrase *the* carries translation cost (-0.916291), distortion or reordering cost (0), language model cost (-2.03434), and word penalty (0). Overall, a probility cost of exp(-2.95064) is accumulated. Together with the future cost estimate for the remaining part if the sentence (-29.4246), this hypothesis is assigned a score of -32.3752.

And so it continues, for a total of 1161 created hypothesis. At the end, the best scoring final hypothesis is found and the hypothesis graph traversed backwards to retrieve the best translation:

```
[ 976 => 583 ]
[ 583 => 106 ]
[ 106 => 12 ]
[ 12 => 0 ]
```

Confused enough yet? Before we get caught too much in the intricate details of the inner workings of the decoder, let us return to actually using the decoder. Much of what has just been said will become much clearer after reading the background information in Section 3.

Parameters
`-v, -verbose` – allows for more detailed output: default (1), summary statistics (2), full progress report (3)

1.4 Tuning for Quality

The key to good translation performance is having a good phrase translation table. But some tuning can be done with the decoder. The most important is the tuning of the model parameters.

The probability cost that is assigned to a translation is a product of probability costs of four models: phrase translation table, language model, reordering model, and word penalty. Each of these models contributes information over one aspect of the characteristics of a good translation:

- The **phrase translation** table ensures that the English phrases and the German phrases are good translations of each other.
- The **language model** ensures that the output is fluent English.
- The **distortion model** allows for reordering of the input sentence, but at a cost: The more reordering, the more expensive is the translation.
- The **word penalty** provides means to ensure that the translations do not get too long or too short.

Each of these components can be given a weight that sets its importance. Mathematically, the cost of translation is:

$$p(e|f) = p_\phi(f|e)^{\lambda_\phi} \times p_{\mathrm{LM}}(e)^{\lambda_{\mathrm{LM}}} \times p_{\mathrm{D}}(e,f)^{\lambda_{\mathrm{D}}} \times \omega^{\mathrm{length(e)}\ \lambda_{\mathrm{W(e)}}} \tag{1}$$

The propability $p(e|f)$ of the English translation e given the foreign input f is broken up into four models, phrase translation $p_\phi(f|e)$, language model $p_{\mathrm{LM}}(e)$, distortion model $p_{\mathrm{D}}(e,f)$, and word penalty $\omega^{\mathrm{length(e)}}$. Each of the four model is weighted by a weight λ.

The weighting is provided to the decoder with the four parameters `weight-t`, `weight-l`, `weight-d`, and `weight-w`. The default setting for these weights are 1,1,1, and 0. These are also the values in Figure 1 on page 5.

Setting these weights to the right values can improve translation quality. We already sneaked in one example above. When translating the German sentence *ein haus ist das*, we set the distortion weight to 0.5 to get the right translation:

```
% echo 'ein haus ist das' | bin/pharaoh -f model/pharaoh.ini -d 0.5
this is a house
```

With the default weights, the translation comes out wrong:

```
% echo 'ein haus ist das' | bin/pharaoh -f model/pharaoh.ini
a house is the
```

What is the right weight setting depends on the corpus and the language pair. Ususally, a held out development set is used to optimize the parameter settings. The simplest method here is to try out with a large number of possible settings, and pick what works best. Good values for the weights for phrase translation table (`weight-t`, short `tm`), language model (`weight-l`, short `lm`), and reordering model (`weight-d`, short `d`) are 0.1–1, good values for the word penalty (`weight-w`, short `w`) are -3–3. Negative values for the word penalty favor longer output, positive values favor shorter output.

Parameters
`-tm, -weight-t` – weight of the phrase translation table
`-lm, -weight-l` – weight of the language model
`-d, -weight-d` – weight of the distortion (reordering) model
`-w, -weight-w` – weight of the word penalty

1.5 Tuning for Speed

Let us now look at some additional parameters that help to speed up the decoder. Unfortunately higher speed usually comes at cost of translation quality. The speed-ups are achieved by limiting the search space of the decoder. By cutting out part of the search space, we may not be able to find the best translation anymore.

1.5.1 Translation Table Size

One strategy to limit the search space is by reducing the number of translation options used for each input phrase, i.e. the number of phrase translation table entries that are retrieved. While in the toy example, the translation tables are very small, these can have thousands of entries per phrase in a realistic scenario. If the phrase translation table is learned from real data, it contains a lot of noise. So, we are really interested only in the most probable ones and would like to elimiate the others.

The are two ways to limit the translation table size: by a fixed limit on how many translation options are retrieved for each input phrase, and by a probability threshold, that specifies that the phrase translation probability has to be above some value.

Compare the statistics and the translation output for our toy model, when no translation table limit is used

```
% echo 'das ist ein kleines haus' |
   bin/pharaoh -f model/pharaoh.ini -ttable-limit 1 -v 2
...
```

```
collected 12 translation options
HYP: 557 added, 32 discarded below threshold, 199 pruned, 572 merged.
BEST: this is a small house -28.9234
```

with the statistics and translation output, when a limit of 1 is used

```
% echo 'das ist ein kleines haus' |
   bin/pharaoh -f model/pharaoh.ini -ttable-limit 1 -v 2
...
collected 6 translation options
HYP: 173 added, 0 discarded below threshold, 0 pruned, 120 merged.
BEST: it is a small house -30.3268
```

Reducing the number of translation options to only one per phrase, had a number of effects: (1) Overall only 6 translation options instead of 12 translation options were collected. (2) The number of generated hypothesis fell to 173 from 557, and no hypotheses were pruned out. (3) The translation changed, and the output now has lower probability — exp(-30.3268) vs. exp(-28.9234).

[-ttable-threshold not implemented yet]

Parameters
`-ttable-limit` – number of phrase translations used for each foreign phrase (default 20)
`-ttable-threshold` – minimum probability for a phrase translation entry (default 0)

1.5.2 Hypothesis Stack Size (Beam)

A different way to reduce the search is to reduce the size of hypothesis stacks. For each number of foreign words translated, the decoder keeps a stack of the best (partial) translations. By reducing this stack size the search will be quicker, since less hypotheses are kept at each stage, and therefore less hypotheses are generated. This is explained in more detail in Section 3.

From a user perspective, search speed is linear to the maximum stack size. Compare the following system runs with stack size 1000, 100 (the default), 10, and 1:

```
% echo 'das ist ein kleines haus' \
      | bin/pharaoh -f model/pharaoh.ini -s 1000 -v 2
...
collected 12 translation options
HYP: 634 added, 0 discarded below threshold, 0 pruned, 1306 merged.
BEST: this is a small house -28.9234

% echo 'das ist ein kleines haus' \
      | bin/pharaoh -f model/pharaoh.ini -s 100 -v 2
...
collected 12 translation options
HYP: 557 added, 32 discarded below threshold, 199 pruned, 572 merged.
BEST: this is a small house -28.9234
```

```
% echo 'das ist ein kleines haus' \
      | bin/pharaoh -f model/pharaoh.ini -s 10 -v 2
...
collected 12 translation options
HYP: 101 added, 93 discarded below threshold, 50 pruned, 29 merged.
BEST: this is a small house -28.9234

% echo 'das ist ein kleines haus' \
      | bin/pharaoh -f model/pharaoh.ini -s 1 -v 2
...
collected 12 translation options
HYP: 9 added, 19 discarded below threshold, 4 pruned, 0 merged.
BEST: this is a little house -30.0117
```

Note that the number of hypothesis entered on stacks is getting smaller with the stack size: 634, 557, 101, and 9. With a stack size of 1000, no pruning takes place, which means an exhaustive search. With smaller stack sizes we risk search errors, meaning the generation of translations that score worse than the best translation according to the model.

In this toy example, a worse translation is only generated with a stack size of 1. Again, by worse translation, we mean worse scoring accrosing to our model (exp(-30.0117) vs. exp(-28.9234)). If it is actually a worse translation in terms of translation quality, is another question. However, the task of the decoder is to find the best scoring translation. If worse scoring translations are of better quality, then this is a problem of the model, and should be resolved by better modeling.

[-beam-threshold not implemented yet]

Parameters
`-s, -stack` – maximum size of the beam, i.e., the hypothesis stacks (default 100)
`-b, -beam-threshold` – minimum value of a hypothesis relative to the best hypothesis in a stack (default 0.00001)

1.5.3 Limit on Distortion (Reordering)

[-distortion-limit not implemented yet]

Parameters
`-distortion-limit` – maximum distance between two input phrases that are translated to two neighboring output phrases (default 0, no limit)

1.6 XML Markup

Sometimes we have external knowledge that we want to bring to the decoder. For instance, we might have a better translation system for translating numbers of dates. We would like to plug in these translations into the decoder without changing the model.

To address this, the decoder has an XML markup scheme that allows the specification of translations for parts of the sentence. In its simplest form, we can tell the decoder what to use to translate certain words in the sentence:

```
% echo 'das ist <np english="a cute place">ein kleines haus</np>' \
      | bin/pharaoh -f model/pharaoh.ini
this is a cute place
```

```
% echo 'das ist ein kleines <n english="dwelling">haus</n>' \
      | bin/pharaoh -f model/pharaoh.ini
this is a little dwelling
```

The words have to be surrounded by tags, such as `<np...>` and `</np>`. The name of the tags can be chosen freely. The target output is specified in the opening tag as a parameter value for a parameter that is called for historical reasons `english` (the canonical target language).

Not only one, but multiple translations can be specified, as the following example shows:

```
% echo 'das ist ein kleines <n english="dwelling|place|house">haus</n>' \
      | bin/pharaoh -f model/pharaoh.ini
that is a little place
```

Here, three choices are given: *dwelling, place, house.* The decoder may pick any of them. Effectively, the language model decides what fits best into the sentence context.

But we can also provide probabilities along with these translation choices:

```
% echo 'das ist ein kleines \
      <n english="dwelling|place|house" prob="0.1|0.1|0.8">haus</n>' \
      | bin/pharaoh -f model/pharaoh.ini
that is a small house
```

Here, the given probability distribution assigns probilities to the three choices: *dwelling* (p=0.1), *place* (p=0.1), and *house* (p=0.8). With these probabilities, the decoder choses *house* instead of *place*.

As a final twist, we can allow the decoder to use either translations from the model or the specified translations:

```
% echo 'das ist ein kleines \
      <n english="dwelling|place" prob="0.9|0.1">haus</n>' \
      | bin/pharaoh -f model/pharaoh.ini -bypass-marked
this is a small house
```

The switch `-bypass-marked` gives the decoder a choice between using the specified translations or its own. This choice, again, is ulitmately made by the language model, which takes the sentence context into account.

But we may also have some notion of how much we trust our specified translations opposed to how much we trust the model. This can be expressed in a weight that is factored into the probabilities of the specified translations:

```
% echo 'das ist ein kleines \
      <n english="dwelling|place" prob="0.9|0.1">haus</n>' \
      | bin/pharaoh -f model/pharaoh.ini -bypass-marked -weight-marked 10
this is a little dwelling
```

The weight 10, specified by the switch `-weight-marked`, is multiplied with the specified probabilities. Alternately, 10-fold probabilities may be specified (`prob="9|1"`), since it is not required that these add up to one. The weighting feature is really just there for convenience, so that different weights can be tried easily without changing the input to the decoder.

One last remark: this XML-scheme does not follow standard markup notation. No surrounding tags are neccessary, and the input format still has to be one sentence per line.

Parameters
`-bypass-marked` – allow to bypass the translations specified with the XML markup
`-weight-marked` – weight the specified translations (default 1) relative to model translations

1.7 N-Best List Generation

So far, we were able to see only the 1-best translation, meaning the best translation that the decoder found, as scored by the model. But we might also want to know what was the second choice, how much its score differs, where the correct translation is ranked, and so on.

The switch `-lattice` lets the decoder output a word lattice. This lattice can be processed by the finite state toolkit Carmel for n-best lists, scoring of specific translations, and other things. While you can also write your own tools to work with the lattice, it is highly recommended to use that toolkit, which is freely available from the Information Sciences Institute at University of Southern California under this address:

`http://www.isi.edu/licensed-sw/carmel/`

This page also contains instructions how to use the finite state toolkit.

But let us return to our decoder and run it with the lattice switch:

```
echo 'das ist ein kleines haus' | \
      bin/pharaoh -f model/pharaoh.ini -lattice foo
```

The decoder runs as before, but it also creates two files:

```
foo.0000
foo.0000.state
```

The first file, `foo.0000` contains the word lattice, the second file, `foo.0000.state` contains additional information for each state which may come in handy. The file name was specified at the command line (by the argument to the switch, `-lattice foo`), with the sentence number (here, 0000) attached to it. Let us take a look at the first lines of `foo.0000`:

```
1162
(0 (1 "the" 0.0523065))
(0 (2 "it" 0.0056504))
(0 (3 "this" 0.00467583))
(0 (4 "is" 0.00107724))
(0 (5 "'s" 1.67017e-05))
(0 (6 "a" 0.00128125))
(0 (7 "an" 0.0002182))
(0 (8 "small" 1.34764e-06))
(0 (9 "little" 5.53138e-07))
```

The first number (1162) specifies the number of the final state. Then, state transitions are listed, one per line. The first transition states that we can move from state 0 (the start state) to state 1 by producing the word *the* with a cost of 0.0523065.

Here are the state transitions that are used in the best path:

```
(0 (12 "this is" 0.014086))
(12 (106 "a" 0.188447))
(106 (583 "small" 0.000706353))
(583 (1162 "house" 1.46468e-07))
```

Before we move on to using Carmel to get all kinds of useful information out of this file, let us take a quick look at the second file. Here are the first lines of `foo.0000.state`:

```
1 10000
2 10000
3 10000
4 01000
5 01000
6 00100
7 00100
8 00010
9 00010
```

This file contains word coverage vectors for each state. For instance, the first state has the coverage vector 10000, meaning that in this state, the first word was already translated (indicated by 1), while the others where not (indicated by 0). One can see, how this file will be useful to retrieve phrase alignments between input and any output that can be found as a path in the word lattice. However, we will not discuss this further in this manual.

The finite state toolkit Carmel has a large number of options, and the reader is advised to take a look at its extensive tutorial. Here, we want to highlight only a few. Let us start with:

```
% carmel -OQWk 5 foo.0000
this is a small house
```

```
this is a little house
it is a small house
this is a small house
it is a small house
```

These are the top 5 translations according to the word lattice. The switch `-k` specifies the number of translations (here, 5). The other switches suppress information. We can drop them to get more verbose output. For instance, we might be interested in the probability scores of each translation:

```
% carmel -OQk 5 foo.0000
this is a small house 2.74626e-13
this is a little house 9.24851e-14
it is a small house 6.74879e-14
this is a small house 3.43281e-14
it is a small house 3.37439e-14
```

Dropping the next switch gives us some insight into what phrase translations were used:

```
% carmel -Ok 5 foo.0000
"this is" "a" "small" "house" 2.74626e-13
"this is" "a" "little" "house" 9.24851e-14
"it is" "a" "small" "house" 6.74879e-14
"this" "is" "a" "small" "house" 3.43281e-14
"it" "is" "a" "small" "house" 3.37439e-14
```

Without any additional switches, we get the state transitions used in each translation (we just do this for the first two here):

```
% carmel -k 2 foo.0000
("this is" :  "this is" / 0.014086 -> 12) ("a" :  "a" / 0.188447 -> 106)
("small" :  "small" / 0.000706353 -> 583) ("house" :  "house" / 1.46468e-07
-> 1162) 2.74626e-13
("this is" :  "this is" / 0.014086 -> 12) ("a" :  "a" / 0.188447 -> 106)
("little" :  "little" / 0.000902867 -> 584) ("house" :  "house" / 3.85897e-08
-> 1162) 9.24851e-14
```

Parameters
`-l, -lattice` – creates word lattice files in a specified location

1.8 Building a Language Model

We provided a language model with the decoder package, which is trained on the Europarl corpus[1]. However, the language model should be trained on a corpus that is suitable to the domain. If the

[1]available at `http://www.isi.edu/~koehn/europarl/`

translation model is trained on a parallel corpus, then the language model should be trained on the same corpus.

Our decoder works with the SRI language modeling toolkit [Stolke, 2002], which is freely available at

```
http://www.speech.sri.com/projects/srilm/
```

The release comes with a program that creates a language model file, as required by our decoder.

A language model can be created by calling:

```
% ngram-count -text test-file -lm language-model-file
```

There are a variety of switches that can be used. The ones that were used to create the Europarl language model were:

```
-interpolate -kndiscount1 -kndiscount2 -kndiscount3
```

1.9 Evaluation

Evaluation the quality of machine translation is not as easy as it seems. We can not simply check if the output the the system matches exactly a reference translation provided by a human translation, since there is too much variation possible. There are many acceptable ways to translate a sentence.

Here are some evaluation metrics that have been recently used for statistical machine translation:

- **Human judgement** is the best way to evaluate machine translation performance. Human judges may be asked, if the output sentence is correct overall, is syntactically well-formed, or carries the same meaning as the original, etc. These can be either hard yes/no decisions or graded an a scale. Two human judges will disagree on some sentences, no matter what the question is, but especially when assigning fine grain grades. A bigger problem with this type of evaluation is, however, that it is very labor intensive. Judging thousands of sentences (a typical test corpus size) implies many hours of work.

- **Word error rate** is a metric that is commonly used in speech recognition. The system output is compared with a reference translation, and the number of correct words is counted. The words have to be either in the same sequence as in the reference translation, or may be reordered (then called *position independent word error rate*).

- **BLEU** is currently the most commonly used metric in statistical machine translation [Papineni et al., 2001]. It is similar to word error rate in that it compares the system output against a reference translation. However, it compares not only single words, but also word bigrams, trigrams, and so on. Most commonly, n-grams of size up to four are used, but the better the system output the longer n-grams track human judgement. Comparison may be against a single or multiple reference translations.

It is not really hard to implement the BLEU metric. A free toolkit for it (and also the related NIST metric) can be obtained from the following web site:

```
http://www.nist.gov/speech/tests/mt/resources/scoring.htm
```

There is some controversy in the machine translation community regarding automatic scoring methods. Researchers use it as an aid, but are aware that ultimately quality judgements have to come from human judges.

1.10 Advanced Features

We conclude this manual with some additional features of the decoder, which allow for some more sophisticated uses.

1.10.1 Multiple Language Models

The decoder may use more than language model. Usually the language model is trained on one side the parallel corpus. But we may also have a second language model that is trained on a larger amount of monolingual text.

When using mulitple language models, each of them has to be assigned a weight. Here an example, where a second language model file is used, with a lower weight:

```
% bin/pharaoh -f model/pharaoh.ini <in >out \
      -lmodel-file lm/europarl.srilm /somewhere/other.srilm \
      -weight-l 0.7 0.4
```

1.10.2 Multiple Translation Tables

Not only multiple language models, but also multiple translation tables can be used. These have to be stored in the same file, however.

The motivation behind this comes from the reframing of the alignment template approach as a phrase translation method: For each phrase translation, two scores are factored in. Besides a maximum likelihood phrase translation score, also a word translation score, that measures how well the words in the phrase match up. See the discussion by Koehn et al. [2003] for more detail on this.

The translation probabilities have to be provided next to each other in the phrase translation table:

```
this is ||| das ist ||| 0.3 0.232
```

As before, multiple weights have to be given to the decoder:

```
% bin/pharaoh -f model/pharaoh.ini <in >out -weight-t 1 0.25
```

1.10.3 Rescoring

This feature was developed to get the scores from the different model components. This is neccessary to do optimize the model weight parameters using methods such as maximum entropy or minimum error rate training [Och and Ney, 2002; Och, 2003].

Since this is done in conjunction with n-best list generation, the input format is related. Recall that we generated a word lattice file for a single input sentence. We now also have to create a rescoring file for each input sentence.

Recall the example from Section 1.7 on page 14, where we created a 5-best list of translations:

```
% carmel -OQWk 5 foo.0000
this is a small house
this is a little house
it is a small house
this is a small house
it is a small house
```

The full details for the first two translations are:

```
% carmel -k 2 foo.0000
("this is" :  "this is" / 0.014086 -> 12) ("a" :  "a" / 0.188447 -> 106)
("small" :  "small" / 0.000706353 -> 583) ("house" :  "house" / 1.46468e-07
-> 1162) 2.74626e-13
("this is" :  "this is" / 0.014086 -> 12) ("a" :  "a" / 0.188447 -> 106)
("little" :  "little" / 0.000902867 -> 584) ("house" :  "house" / 3.85897e-08
-> 1162) 9.24851e-14
```

This information has to augmented with reordering information and reformulated into:

```
   E this is F das ist D 0 P 0.014086 E a F ein D 0 P 0.188447 E small F kleines
\
      D 0 P 0.000706353 E house F haus D 0 P 1.46468e-07 T 2.74626e-13
   E this is F das ist D 0 P 0.014086 E a F ein D 0 P 0.188447 E little F kleines
\
      D 0 P 0.000902867 E house F haus D 0 P 3.85897e-08 T 9.24851e-14
```

The format of this file is as follows: Each translation is listed in a separate line. Each line consists of the phrase translations that make up the translation and the final total score (e.g., `T 2.74626e-13`). For each phrase translation four things have to be specified: the target translation (e.g., `E this is`), the source (e.g., `F das ist`), the distortion (e.g., `D 0`), and the probability for this step (e.g., `P 0.014086`). The probability specifications (`T`, `P`) may be filled by dummy values, since they are only used for debugging purposes.

With this file, we can call the decoder:

```
% bin/pharaoh -f model/pharaoh.ini -rescore foo.best2.0000
```

The output is scored in the file `foo.best2.0000.rescore`:

```
pD: 0, pLM[0]:  -27.0908, pTM: -1.83258, pWP: -5, reported score:  2.74626e-13
pD: 0, pLM[0]:  -28.1791, pTM: -1.83258, pWP: -5, reported score:  9.24851e-14
```

The file lists the scores for the different components: distortion (`pD`), language model (`pLM[0]`), translation model (`pTM`) and word penalty (`pWP`). These scores are unweighted, meaning that the weight paramaters have not been factored in.

To check, that the scores are correct, we can compute:

$$\exp(0)^1 \times \exp(-27.0908)^1 \times \exp(-1.83258)^1 \times \exp(-5)^0 = 2.746e - 13 \tag{2}$$

If we want to rescore multiple files, there is a convenient option for that as well. The file stem has to be provided as well as the number of files, which is quote similar to the word lattice creation. Here is the command that rescores one file with the file stem `foo.best2`, effectively doing the same as before:

```
% bin/pharaoh -f model/pharaoh.ini -rescoredir foo.best2 1
```

Parameters
`-r, -rescore` – rescore a given translation with the model components
`-rd, -rescoredir` – rescore multiple files

2 Phrase-Based Statistical Machine Translation

Statistical Machine Translation as a research area started in the late 1980s with the Candide project at IBM [Brown et al., 1990]. IBM's original approach maps individual words to words and allows for deletion and insertion of words.

Lately, various researchers have shown better translation quality with the use of phrase translation. Phrase-based MT can be traced back to Och [1998]'s alignment template model, which can be re-framed as a phrase translation system. Other researchers used augmented their systems with phrase translation, such as Yamada and Knight [2001], who use phrase translation in a syntax-based.

Marcu and Wong [2002] introduced a joint-probability model for phrase translation. At this point, most competitive statistical machine translation systems use phrase translation, such as the CMU [Vogel et al., 2003] and IBM [Tillmann, 2003] systems. Phrase-based systems came out ahead at a recent international machine translation competition (DARPA TIDES Machine Translation Evaluation 2003 on Chinese-English and Arabic-English).

Of course, there are other ways to do machine translation. Most commercial systems use transfer rules and a rich translation lexicon. Until recently, machine translation research has been focused on knowledge based systems that use a interlingua representation as an intermediate step between input and output.

There are also other ways to do statistical machine translation. There is some effort in building syntax-based models that either use real syntax trees generated by syntactic parsers [Yamada and Knight, 2001; Koehn and Knight, 2002; Schafer and Yarowski, 2003], or tree transfer methods motivated by syntactic reordering patterns [Wu, 1997; Alshawi et al., 2000].

The phrase-based statistical machine translation model we present here was defined by Koehn et al. [2003]. See also the description by Zens et al. [2002]. The alternative phrase-based methods differ in the way the phrase translation table is created, which we disuss in detail below.

2.1 Model

Figure 2 on the following page illustrates the process of phrase-based translation. The input is segmented into a number of sequences of consecutive words (so-called "phrases"). Each phrase is translated into an English phrase, and English phrases in the output may be reordered.

In this section, we will define the phrase-based machine translation model formally. The phrase translation model is based on the noisy channel model. We use Bayes rule to reformulate the translation probability for translating a foreign sentence $\mathbf{f}$ into English $\mathbf{e}$ as

$$\text{argmax}_{\mathbf{e}} p(\mathbf{e}|\mathbf{f}) = \text{argmax}_{\mathbf{e}} p(\mathbf{f}|\mathbf{e}) p(\mathbf{e})$$

This allows for a language model $p(\mathbf{e})$ and a separate translation model $p(\mathbf{f}|\mathbf{e})$.

During decoding, the foreign input sentence $\mathbf{f}$ is segmented into a sequence of I phrases $\bar{f}_1^I$. We assume a uniform probability distribution over all possible segmentations.

Each foreign phrase $\bar{f}_i$ in $\bar{f}_1^I$ is translated into an English phrase $\bar{e}_i$. The English phrases may be reordered. Phrase translation is modeled by a probability distribution $\phi(\bar{f}_i|\bar{e}_i)$. Recall that due to the Bayes rule, the translation direction is inverted from a modeling standpoint.

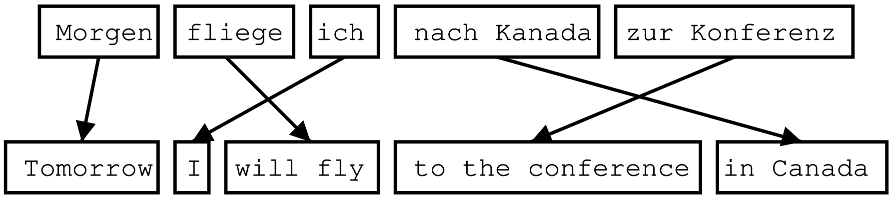


Figure 2: Phrase-based machine translation: input is segmented in phrases, each is translated and may be reordered.

Reordering of the English output phrases is modeled by a relative distortion probability distribution $d(a_i - b_{i-1})$, where a_i denotes the start position of the foreign phrase that was translated into the ith English phrase, and b_{i-1} denotes the end position of the foreign phrase that was translated into the $(i-1)$th English phrase.

We use a simple distortion model $d(a_i - b_{i-1}) = \alpha^{|a_i - b_{i-1} - 1|}$ with an appropriate value for the parameter α.

In order to calibrate the output length, we introduce a factor ω (called word cost) for each generated English word in addition to the trigram language model p_{LM}. This is a simple means to optimize performance. Usually, this factor is larger than 1, biasing toward longer output.

In summary, the best English output sentence $\mathbf{e}_{\mathrm{best}}$ given a foreign input sentence $\mathbf{f}$ according to our model is

$$\begin{aligned}\mathbf{e}_{\mathrm{best}} &= \mathrm{argmax}_{\mathbf{e}} p(\mathbf{e}|\mathbf{f}) \\ &= \mathrm{argmax}_{\mathbf{e}} p(\mathbf{f}|\mathbf{e}) p_{\mathrm{LM}}(\mathbf{e}) \omega^{\mathrm{length}(\mathbf{e})}\end{aligned}$$

where $p(f|e)$ is decomposed into

$$p(\bar{f}_1^I|\bar{e}_1^I) = \prod_{i=1}^{I} \phi(\bar{f}_i|\bar{e}_i) d(a_i - b_{i-1})$$

2.2 Word Alignment

When describing the phrase-based translation model so far, we did not discuss how to obtain the model parameters, especially the phrase probability translation table that maps foreign phrases to English phrases.

Most recently published methods on extracting a phrase translation table from a parallel corpus start with a word alignment. Word alignment is an active research topic. For instance, this problem was the focus as a shared task at a recent data driven machine translation workshop [Mihalcea and Pedersen, 2003]. See also the systematic comparison by Och and Ney [2003].

At this point, the most common tool to establish a word alignment is to use the toolkit Giza++[2]. This toolkit is an implementation of the original IBM Models that started statistical machine translation research. However, these models have some serious draw-backs. Most importantly, they only allow at

[2]available at `http://www.isi.edu/~och/GIZA++.html`

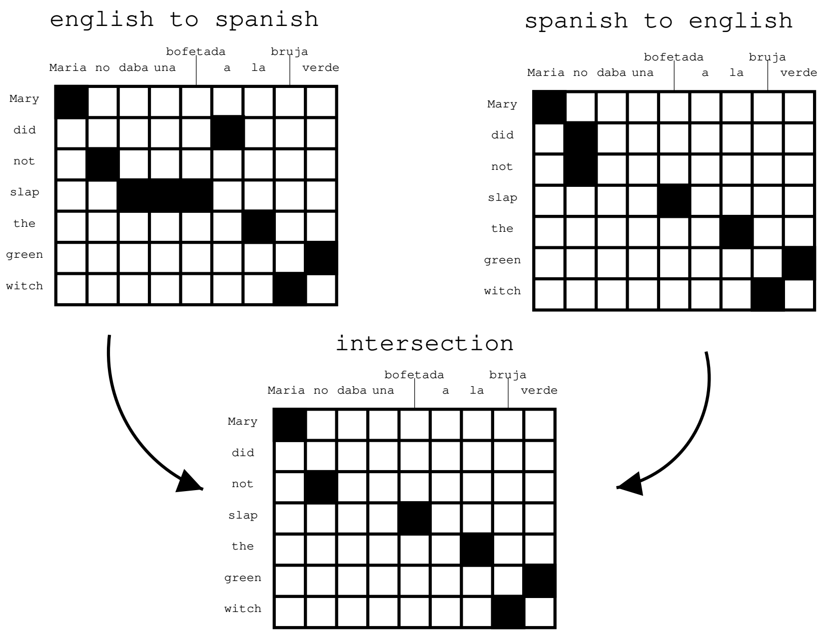


Figure 3: Intersection of two alignments produced by the Giza++ toolkit, a common strategy to obtain high-precision word alignments

most one English word to be aligned with each foreign word. To resolve this, some transformations are applied.

First, the parallel corpus is aligned bidirectionally, e.g., Spanish to English and English to Spanish. This generates two word alignments that have to be reconciled. If we intersect the two alignments, we get a high-precision alignment of high-confidence alignment points. If we take the union of the two alignments, we get a high-recall alignment with additional alignment points. See Figure 3 for an illustration.

Researchers differ in their methods where to go from here. We describe the details below.

2.3 Methods for Learning Phrase Translations

Most of the recently proposed methods use a word alignment to learn a phrase translation table. We discuss three such methods in this section [Och, 2002; Venugopal et al., 2003; Tillmann, 2003], and one exception.

2.3.1 Marcu and Wong

First, the exception: Marcu and Wong [2002] proposed to establish phrase correspondences directly in a parallel corpus. To learn such correspondences, they introduced a phrase-based joint probability model that simultaneously generates both the source and target sentences in a parallel corpus.

Expectation Maximization learning in Marcu and Wong's framework yields both (i) a joint probability distribution $\phi(\bar{e}, \bar{f})$, which reflects the probability that phrases $\bar{e}$ and $\bar{f}$ are translation equivalents; (ii) and a joint distribution $d(i, j)$, which reflects the probability that a phrase at position i is translated into

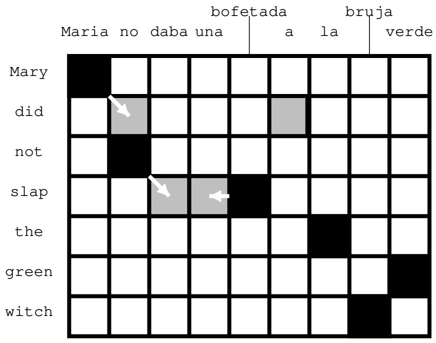


Figure 4: Illustration of the refinement heuristics by Och and Ney [2003]: Points from the intersection (black) are expanded (arrows) into points of the union (grey).

a phrase at position j.

To use this model in the context of our framework, we simply marginalize the joint probabilities estimated by Marcu and Wong [2002] to conditional probabilities. Note that this approach is consistent with the approach taken by Marcu and Wong themselves, who use conditional models during decoding.

2.3.2 Och et al.

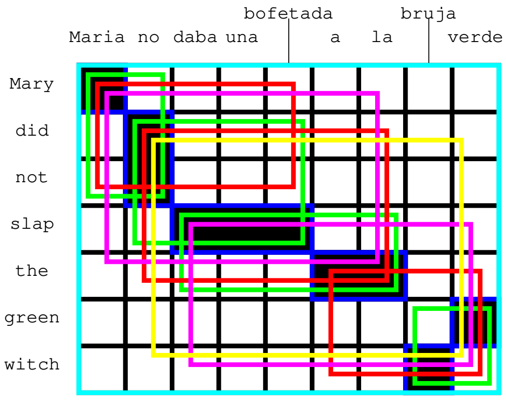
Och and Ney [2003] propose a heuristic approach to refine the alignments obtained from Giza++. At a minimum, all alignment points of the intersection of the two alignments are maintained. At a maximum, the points of the union of the two alignments are considered. To illustrate this, see Figure 4. The intersection points are black, the additional points in the union are shaded grey.

Och and Ney explore the space between intersection and union with expansion heuristics that start with the intersection and add additional alignment points. The decision which points to add may depend on a number of criteria:

- In which alignment does the potential alignment point exist? Foreign-English or English-foreign?
- Does the potential point neighbor already established points?
- Does "neighboring" mean directly adjacent (block-distance), or also diagonally adjacent?
- Is the English or the foreign word that the potential point connects unaligned so far? Are both unaligned?
- What is the lexical probability for the potential point?

Och and Ney [2003] are ambigous in their description about which alignment points are added in their refined method. We reimplemented their method recently [Koehn et al., 2003], so we will describe this interpretation here.

Our heuristic proceeds as follows: We start with intersection of the two word alignments. We only add new alignment points that exist in the union of two word alignments. We also always require that a new alignment point connects at least one previously unaligned word.




```
(Maria, Mary), (no, did not), (slap, daba una bofetada), (a la, the), (bruja, witch),
(verde, green), (Maria no, Mary did not), (no daba una bofetada, did not slap),
(daba una bofetada a la, slap the), (bruja verde, green witch),
(Maria no daba una bofetada, Mary did not slap),
(no daba una bofetada a la, did not slap the), (a la bruja verde, the green witch)
(Maria no daba una bofetada a la, Mary did not slap the),
(daba una bofetada a la bruja verde, slap the green witch),
(no daba una bofetada a la bruja verde, did not slap the green witch),
(Maria no daba una bofetada a la bruja verde, Mary did not slap the green witch)
```

Figure 5: Learning all phrase pairs that are consistent with the word alignment

First, we expand to only directly adjacent alignment points. We check for potential points starting from the top right corner of the alignment matrix, checking for alignment points for the first English word, then continue with alignment points for the second English word, and so on.

This is done iteratively until no alignment point can be added anymore. In a final step, we add non-adjacent alignment points, with otherwise the same requirements.

We collect all aligned phrase pairs that are consistent with the word alignment: The words in a legal phrase pair are only aligned to each other, and not to words outside. The set of bilingual phrases **BP** can be defined formally [Zens et al., 2002] as:

$$\mathbf{BP}(f_1^J, e_1^J, A) = \{(f_j^{j+m}, e_i^{i+n}) : \forall (i', j') \in A : j \leq j' \leq j+m \leftrightarrow i \leq i' \leq i+n\} \tag{3}$$

Figure 5 displays all the phrase pairs that are collected according to this definition for the alignment from our running example.

Given the collected phrase pairs, we estimate the phrase translation probability distribution by relative frequency:

$$\phi(\bar{f}|\bar{e}) = \frac{\text{count}(\bar{f}, \bar{e})}{\sum_{\bar{f}} \text{count}(\bar{f}, \bar{e})}$$

No smoothing is performed, although lexical weighting addresses the problem of sparse data. For more details, see our paper on phrase-based translation [Koehn et al., 2003].

2.3.3 Tillmann

Tillmann [2003] proposes a variation of this method. He starts with phrase alignments based on the intersection of the two Giza alignments and uses points of the union to expand these. See his presentation for details.

2.3.4 Venugopal, Zhang, and Vogel

Venugopal et al. [2003] allows also for the collection of phrase pairs that are violated by the word alignment. They introduce a number of scoring methods take consistency with the word alignment, lexical translation probabilities, phrase length, etc. into account.

Zhang et al. [2003] proposes a phrase alignment method that is based on word alignments and tries to find a unique segmentation of the sentence pair, as it is done by Marcu and Wong [2002] directly. This enables them to estimate joint probability distributions, which can be marginalized into conditional probability distributions.

Vogel et al. [2003] reviews these two methods and shows that the combining phrase tables generated by different methods improves results.

3 Decoder

This section describes the decoder Pharaoh from a more theoretical perspective. The decoder was originally developed for the phrase model proposed by Marcu and Wong [2002]. At that time, only a greedy hill-climbing decoder was available, which was unsufficent for our work on noun phrase translation [Koehn, 2003].

The decoder implements a beam search and is roughly similar to work by Tillmann [2001] and Och [2002]. In fact, by reframing Och's alignment template model as a phrase translation model, the decoder is also suitable for his model, as well as other recently proposed phrase models [Venugopal et al., 2003; Tillmann, 2003].

We start this section with defining the concept of translation options, describe the basic mechanism of beam search, and its neccessary components: pruning, future cost estimates. We conclude with background on n-best list generation and the XML implementation.

3.1 Translation Options

Given an input string of words, a number of phrase translations could be applied. We call each such applicable phrase translation a *translation option*. This is illustrated in Figure 6, where a number of phrase translations for the Spanish input sentence *Maria no daba uma bofetada a la bruja verde* are given.

These translation options are collected before any decoding takes place. This allows a quicker lookup than consulting the whole phrase translation table during decoding. The translation options are stored with the information

- first foreign word covered
- last foreign word covered
- English phrase translation
- phrase translation probability

Note that only the translation options that can be applied to a given input text are necessary for decoding. Since the entire phrase translation table may be to big to fit into memory, we can restrict ourselves to these translation options to overcome such computational concerns. We may even generate

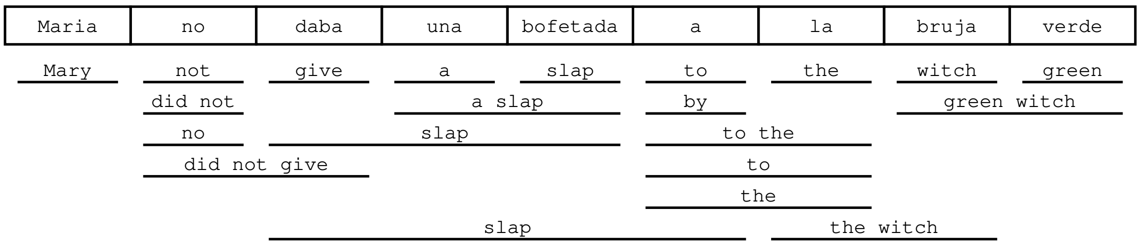


Figure 6: Some translation options for the Spanish input sentence *Maria no daba una bofetada a la bruja verde*

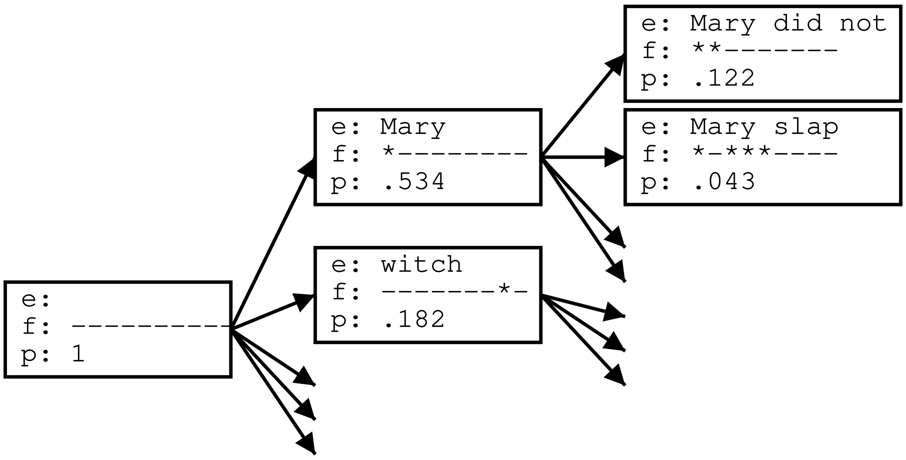


Figure 7: State expansion in the beam decoder: in each expansion English words are generated, additional foreign words are covered (marked by *), and the probability cost so far is adjusted. In this example the input sentence is *Maria no daba una bofetada a la bruja verde.*

a phrase translation table on demand that only includes valid translation options for a given input text. This way, a full phrase translation table (that may be computationally too expensive to produce) may never have to be built.

3.2 Core Algorithm

The phrase-based decoder we developed employs a beam search algorithm, similar to the one used by [Jelinek, 1998, Chapter 5] for speech recognition. The English output sentence is generated left to right in form of hypotheses.

This process illustrated in Figure 7. Starting from the initial hypothesis, the first expansion is the foreign word *Maria*, which is translated as *Mary*. The foreign word is marked as translated (marked by an asterisk). We may also expand the initial hypothesis by translating the foreign word *bruja* as *witch*.

We can generate new hypotheses from these expanded hypotheses. Given the first expanded hypothesis we generate a new hypothesis by translating *no* with *did not*. Now the first two foreign words *Maria* and *no* are marked as being covered. Following the back pointers of the hypotheses we can read of the (partial) translations of the sentence.

Let us now describe the beam search more formally. We begin the search in an initial state where no foreign input words are translated and no English output words have been generated. New states are created by extending the English output with a phrasal translation of that covers some of the foreign input words not yet translated.

The current cost of the new state is the cost of the original state multiplied with the translation, distortion and language model costs of the added phrasal translation. Note that we use the informal concept *cost* analogous to probability: A high cost is a low probability.

Each search state (hypothesis) is represented by

- a back link to the best previous state (needed for find the best translation of the sentence by back-tracking through the search states)

- the foreign words covered so far
- the last two English words generated (needed for computing future language model costs)
- the end of the last foreign phrase covered (needed for computing future distortion costs)
- the last added English phrase (needed for reading the translation from a path of hypotheses)
- the cost so far
- an estimate of the future cost (is precomputed and stored for efficiency reasons, as detailed in Section 3.5)

Final states in the search are hypotheses that cover all foreign words. Among these the hypothesis with the lowest cost (highest probability) is selected as best translation.

The algorithm described so far can be used for exhaustively searching through all possible translations. In the next sections we will describe how to optimize the search by discarding hypotheses that cannot be part of the path to the best translation. We then introduce the concept of comparable states that allow us to define a beam of good hypotheses and prune out hypotheses that fall out of this beam. In a later section (Section 3.6), we will describe how to generate an (approximate) n-best list.

3.3 Recombining Hypotheses

Recombining hypothesis is a risk-free way to reduce the search space. Two hypotheses can be recombined if they agree in

- the foreign words covered so far
- the last two English words generated
- the end of the last foreign phrase covered

If there are two paths that lead to two hypotheses that agree in these properties, we keep only the cheaper hypothesis, e.g., the one with the least cost so far. The other hypothesis cannot be part of the path to the best translation, and we can safely discard it.

Note that the inferior hypothesis can be part of the path to the second best translation. This is important for generating n-best lists. We return to this point in Section 3.6.

3.4 Beam Search

While the recombination of hypotheses as described above reduces the size of the search space, this is not enough for all but the shortest sentences. Let us estimate how many hypotheses (or, states) are generated during an exhaustive search. Considering the possible values for the properties of unique hypotheses, we can estimate an upper bound for the number of states by

$$N \simeq 2^{n_f} |V_e|^2 n_f \tag{4}$$

where n_f is the number of foreign words, and $|V_e|$ the size of the English vocabulary. In practice, the number of possible English words for the last two words generated is much smaller than $|V_e|^2$. The main concern is the exponential explosion from the 2^{n_f} possible configurations of foreign words covered by a hypothesis. Note this causes the problem of machine translation to become NP-complete [Knight, 1999] and thus dramatically harder than, for instance, speech recognition.

In our beam search we compare the hypotheses that cover the same *number* of foreign words and prune out the inferior hypotheses. We could base the judgment of what inferior hypotheses are on the cost of each hypothesis so far. However, this is generally a very bad criterion, since it biases the search to first translating the easy part of the sentence. For instance, if there is a three word foreign phrase that easily translates into a common English phrase, this may carry much less cost than translating three words separately into uncommon English words. The search will prefer to start the sentence with the easy part and discount alternatives too early.

So, our measure for pruning out hypotheses in our beam search does not only include the cost so far, but also an estimate of the future cost. This future cost estimation should favor hypotheses that already covered difficult parts of the sentence and have only easy parts left, and discount hypotheses that covered the easy parts first. We describe the details of our future cost estimation in the next section.

Given the cost so far and the future cost estimation, we can prune out hypotheses that fall outside the beam. The beam size can be defined by threshold and histogram pruning. A relative threshold cuts out a hypothesis with a probability less than a factor α of the best hypotheses (e.g., $\alpha = 0.001$). Histogram pruning keeps a certain number n of hypotheses (e.g., $n = 1000$).

Note that this type of pruning is not risk-free (opposed to the recombination, which we described earlier in Section 3.3). If the future cost estimates are inadequate, we may prune out hypotheses on the path to the best scoring translation. In a particular version of beam search, A* search, the future cost estimate is required to be *admissible*, which means that it never overestimates the future cost. Using best-first search and an admissible heuristic allows pruning that is risk-free. In practice, however, this type of pruning does not sufficiently reduce the search space. See more on search in any good Artificial Intelligence text book, such as the one by Russel and Norvig [1995].

Figure 8 describes the algorithm we used for our beam search. For each number of foreign words covered, a hypothesis stack in created. The initial hypothesis is placed in the stack for hypotheses with no foreign words covered. Starting with this hypothesis, new hypotheses are generated by committing to phrasal translations that covered previously unused foreign words. Each derived hypothesis is placed in a stack based on the number of foreign words it covers.

We proceed through these hypothesis stacks, going through each hypothesis in the stack, deriving new hypotheses for this hypothesis and placing them into the appropriate stack (see Figure 9 for an illustration). After a new hypothesis is placed into a stack, the stack may have to be pruned by threshold or histogram pruning, if it has become too large. In the end, the best hypothesis of the ones that cover all foreign words is the final state of the best translation. We can read off the English words of the translation by following the back links in each hypothesis.

```
initialize hypothesisStack[0 .. nf];
create initial hypothesis hyp_init;
add to stack hypothesisStack[0];
for i=0 to nf-1:
  for each hyp in hypothesisStack[i]:
    for each new_hyp that can be derived from hyp:
      nf[new_hyp] = number of foreign words covered by new_hyp;
      add new_hyp to hypothesisStack[nf[new_hyp]];
      prune hypothesisStack[nf[new_hyp]];
find best hypothesis best_hyp in hypothesisStack[nf];
output best path that leads to best_hyp;
```

Figure 8: Pseudo code for the beam search algorithm

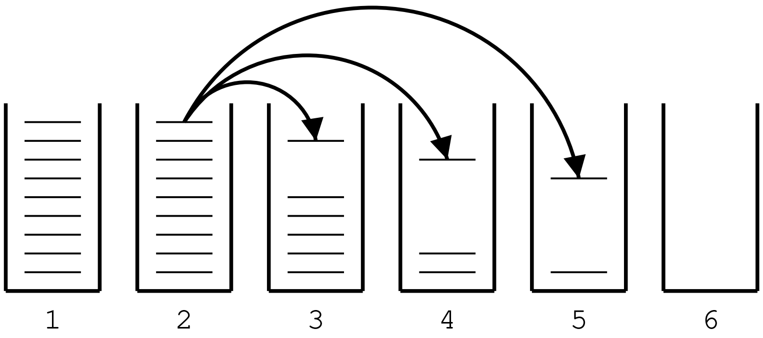


Figure 9: Hypothesis expansion: Hypotheses are placed in stacks according to the number of foreign words translated so far. If a hypothesis is expanded into new hypotheses, these are placed in new stacks.

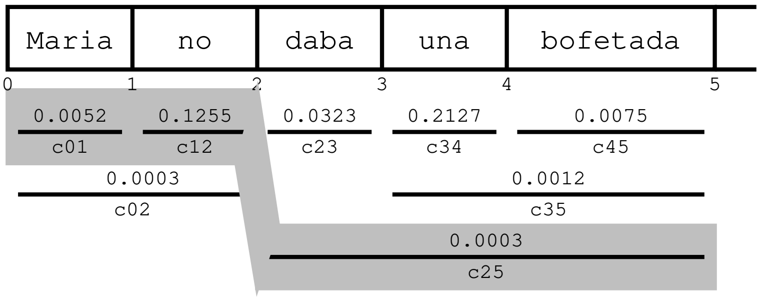


Figure 10: Finding the best future cost path through translation options. The cheapest cost is $c_{01}c_{12}c_{25} = 0.0052 \times 0.1255 \times 0.0003 = 1.9578 \times 10^{-7}$, hence it is the estimate of the cost of translating the five words *Maria no daba una bofetada.*

3.5 Future Cost Estimation

Recall that for excluding hypotheses from the beam we do not only have to consider the cost so far, but also an estimate of the future cost. While it is possible to calculate the cheapest possible future cost for each hypothesis, this is computationally so expensive that it would defeat the purpose of the beam search.

The future cost is tied to the foreign words that are not yet translated. In the framework of the phrase-based model, not only may single words be translated individually, but also consecutive sequences of words as a phrase.

Each such translation operation carries a translation cost, language model costs, and a distortion cost. For our future cost estimate we consider only translation and language model costs. The language model cost is usually calculated by a trigram language model. However, we do not know the preceding English words for a translation operation. Therefore, we approximate this cost by computing the language model score for the generated English words alone. That means, if only one English word is generated, we take its unigram probability. If two words are generated, we take the unigram probability of the first word and the bigram probability of the second word, and so on.

For a sequence of foreign words multiple overlapping translation options exist. We just described how we calculate the cost for each translation option. The cheapest way to translate the sequence of foreign words includes the cheapest translation options. We approximate the cost for a path through translation options by the product of the cost for each option.

To illustrate this concept, refer to Figure 10. The translation options cover different consecutive foreign words and carry an estimated cost c_{ij}. The cost of the shaded path through the sequence of translation options is $c_{01}c_{12}c_{25} = 1.9578 \times 10^{-7}$.

The cheapest path for a sequence of foreign words can be quickly computed with dynamic programming. Also note that if the foreign words not covered so far are two (or more) disconnected sequences of foreign words, the combined cost is simply the product of the costs for each contiguous sequence. Since there are only $n(n+1)/2$ contiguous sequences for n words, the future cost estimates for these sequences can be easily precomputed and cached for each input sentence. Looking up the future costs for a hypothesis can then be done very quickly by table lookup. This has considerable speed advantages over computing future cost on the fly.

3.6 N-Best Lists Generation

Usually, we expect the decoder to give us the best translation for a given input according to the model. But for some applications, we might also be interested in the second best translation, third best translation, and so on.

A common method in speech recognition, that has also emerged in machine translation [Koehn and Knight, 2003; Och et al., 2003] is to first use a machine translation system such as our decoder as a base model to generate a set of candidate translations for each input sentence. Then, additional features are used to rescore these translations.

An n-best list is one way to represent multiple candidate translations. Such a set of possible translations can also be represented by word graphs [Ueffing et al., 2002] or forest structures over parse trees [Langkilde, 2000]. These alternative data structures allow for more compact representation of a much larger set of candidates. However, it is much harder to detect and score global properties over such data structures.

3.6.1 Additional Arcs in the Search Graph

Recall the process of state expansions, illustrated in Figure 7. The generated hypotheses and the expansions that link them form a graph. Paths branch out when there are multiple translation options for a hypothesis from which multiple new hypotheses can be derived. Paths join when hypotheses are recombined.

Usually, when we recombine hypotheses, as described in Section 3.3, we simply discard the worse hypothesis, since it cannot possibly be part of the best path through the search graph (in other words, part of the best translation).

But since we are now also interested in the second best translation, we cannot simply discard information about that hypothesis. If we would do this, the search graph would only contain one path for each hypothesis in the last hypothesis stack (which contains hypotheses that cover all foreign words).

If we store information that there are multiple ways to reach a hypothesis, the number of possible paths also multiplies along the path when we traverse backward through the graph.

In order to keep the information about merging paths, we keep a record of such merges that contains

- identifier of the previous hypothesis
- identifier of the lower-cost hypothesis
- cost from the previous to higher-cost hypothesis

Figure 11 gives an example for the generation of such an arc: in this case, the hypotheses 2 and 4 are equivalent in respect to the heuristic search, as detailed in Section 3.3. Hence, hypothesis 4 is deleted. But since we want keep the information about the path leading from hypothesis 3 to 2, we store a record of this arc. The arc also contains the cost added from hypothesis 3 to 4. Note that the cost from hypothesis 1 to hypothesis 2 does not have to be stored, since it can be recomputed from the hypothesis data structures.

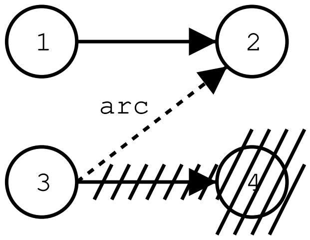


Figure 11: Keeping a record of an arc for n-best list generation: if hypothesis 2 and 4 are equivalent with respect to the heuristic search, hypothesis 4 is deleted (hypothesis recombination), but a record of the arc(3, 2, $\text{cost}_4 - \text{cost}_3$) is kept.

3.6.2 Mining the Search Graph for an n-Best List

The graph of the hypothesis space can be also be viewed as a probabilistic finite state automaton. The hypotheses are states, and the records of back-links and the additionally stored arcs are state transitions. The added probability scores when expanding a hypothesis are the costs of the state transitions.

Finding the n-best path in such a probabilistic finite state automaton is a well-studied problem. In our implementation, we store the information about hypotheses, hypothesis transitions, and additional arcs in a file that can be processed by the finite state toolkit Carmel[3], which we use to mine the n-best lists. This toolkit uses the n shortest paths algorithm by Eppstein [1994].

Our method is related to work by Ueffing et al. [2002] for generating n-best lists for IBM Model 4.

3.7 XML-Markup

While statistical machine translation methods cope well with many aspects of translation of languages, there are a few special problems for which better solutions exist. One example is the translation of named entity, such as proper names, dates, quantities, and numbers.

Consider the task of translating numbers, such as *17.55*. In order for a statistical machine translation system to be able to translate this number, it has to observed it in the training data. But even if it has been seen a few times, it is possible that the translation table learned for this “word” is very noisy.

Translating numbers is not a hard problem. It is therefore desirable to be able to tell the decoder up front how to translate such numbers.

The translation of named entities, for which other methods exist, was the motivation for the work by Germann [2003] to develop a XML markup scheme that, among other things, allows the specification of predefined translations in the input to a statistical machine translation system.

To continue our example of the number 17.55, this may take the following form:

```
Er erzielte <NUMBER english='17.55'>17,55</NUMBER> Punkte .
```

This modified input passes to the decoder not only the German words, but also that the third word, the number *17,55*, should be translated as *17.55*. The decoder accepts this translation and does not try to find a translation on its own.

[3]available at `http://www.isi.edu/licensed-sw/carmel/`

The use of such predefined translations for named entities has been shown to improve the quality of a statistical machine translation system significantly, especially when only a small parallel corpus is available.

We extended the XML scheme for our work on noun phrase translation [Koehn, 2003]. Here, we separate out the noun phrases of a setence, translate them separately, and plug the translations back into a full sentence translation system.

We provide these translation using the same XML markup scheme:

```
Es ist <NPPP english='a small house'>ein kleines Haus</NPPP> .
```

Here, the noun phrase *ein kleines Haus* was recognized in the input. It was passed to the NP/PP subsystem, which translated it as *a small house*. This translation is now specified in the modified input. This instructs the decoder to use this as a translation for this part of the sentence.

3.7.1 Phrase-Based Translation with XML Markup

Germann [2003] developed an implementation of the XML markup scheme for a greedy decoder for a word-based statistical machine translation model. We implemented and extended it for a beam search decoder for a phrase-based statistical machine translation model.

Marking a sequence of words and specifying a translation for them fits neatly into the framework of phrase-based translation. In a way, for a given phrase in the sentence, a translation is provided, which is in essence a phrase translation with translation probability 1. Only for the other parts of the sentence, translation options are generated from the phrase translation table.

Since the first step of the implementation of the beam search decoder is to collect all possible translation options, only this step has to be altered to be sensitive to specifications via XML markup. The core algorithm remains unchanged.

3.7.2 Passing Probability Distribution of Translations

Making the hard decision of specifying the translation for parts of the sentence has some draw-backs. For instance, the number *1* may be translated as *1, one, a,* and so on into English. So we may want to provide a set of possible translations.

Instead of passing one best translation choice to the decoder, we may want to pass along a probability distribution over a set of possible translations. Given several choices, the decoder is aided by the language model to sort out which translation to use.

We extend the XML markup scheme by allowing the specification of multiple English translation options along with translation probabilities.

To give an example:

```
Es ist <NPPP english='a small house|a little house' prob='0.6|0.4'>ein kleines Haus</NPPP>
```

Here, both *a small house* and *a little house* are passed along as possible translations, with the translation probabilities 0.6 and 0.4, respectively.

The scores that are passed along with the translations do not have to be probabilities in a strict sense, e.g., they do not have to add up to 1. They also do not include language model probabilities, but only the factor $p(f|e)$ for the marked part of the sentence.

As in the case of passing on a single best translation, we treat the passed translations as translation options in the decoder. Now, they also include translation probability score, which corresponds to a phrase translation probability score.

The language model scores for all the words in the transfered translation and the phrase translations chosen by the decoder are factored in. This way, the language model helps with the selection of the best translation in the passed probability distribution, using the actual sentence context in which it is used.

3.7.3 Multi-Path Integration

By specifying a set of possible translations, we can deal with uncertainty which of the translations is the right one. But there is also the uncertainty, when to use specified translations at all. Maybe the original model has a better way to deal with the targeted words.

Recognizing that the hard decision of breaking out certain words in the input sentence and providing translations to them may be occasionally harmful, we now want to relax this decision. We allow the decoder to use the specified translations, but also to bypass them and use its own translations.

We call this multi-path integration, since we allow two pathways when translating. The path may go through the specified translations, or we use translation options from the regular translation model.

Recall that the specified translations are in the same format as the regular translation options from the phrase translation table. This made the implementation of the XML markup scheme so straight-forward. Passing on an n-best of translations to the decoder is the same as looking up the phrase translation for a given phrase from the phrase translation table. In both cases, possible translations for a part of the input sentence with probabilities are provided to the decoder.

In other words: In multi-path translation, not only all the translation options from the phrase translation table can be used for translation any part of the sentence (including marked up words), but also the specified translations. It is left to the full sentence translation system to pick which translation option to use.

One may conceive of many different weighting schemes to balance regular phrase translation table entries against specified translations. The decoder allows for scaling the probabilities of the specified translations with a factor.

References

Alshawi, H., Bangalore, S., and Douglas, S. (2000). Learning dependency translation models as collection of finite-state head transducers. *Computational Linguistics*, 26(1):45–60.

Brown, P., Cocke, J., Pietra, S. A. D., Pietra, V. J. D., Jelinek, F., Lafferty, J. D., Mercer, R. L., and Rossin, P. (1990). A statistical approach to machine translation. *Computational Linguistics*, 16(2):76–85.

Eppstein, D. (1994). Finding the k shortest paths. In *Proc. 35th Symp. Foundations of Computer Science*, pages 154–165. IEEE.

Germann, U. (2003). Greedy decoding for statistical machine translation in almost linear time. In *Proceedings of the Joint Conference on Human Language Technologies and the Annual Meeting of the North American Chapter of the Association of Computation Linguistics (HLT-NAACL).*

Jelinek, F. (1998). *Statistical Methods for Speech Recognition.* The MIT Press.

Knight, K. (1999). Decoding complexity in word-replacement translation models. *Computational Linguistics*, 25(4):607–615.

Koehn, P. (2002). Europarl: A multilingual corpus for evaluation of machine translation. Unpublished, `http://www.isi.edu/~koehn/europarl/`.

Koehn, P. (2003). *Noun Phrase Translation.* PhD thesis, University of Southern California, Los Angeles.

Koehn, P. and Knight, K. (2002). ChunkMT: Machine translation with richer linguistic knowledge. Unpublished.

Koehn, P. and Knight, K. (2003). Feature-rich translation of noun phrases. In *41st Annual Meeting of the Association of Computational Linguistics (ACL).*

Koehn, P., Och, F. J., and Marcu, D. (2003). Statistical phrase based translation. In *Proceedings of the Joint Conference on Human Language Technologies and the Annual Meeting of the North American Chapter of the Association of Computational Linguistics (HLT-NAACL).*

Langkilde, I. (2000). Forest-based statistical sentence generation. In *Proceedings of Annual Meeting of the North American Chapter of the Association of Computational Linguistics (NAACL).*

Marcu, D. and Wong, W. (2002). A phrase-based, joint probability model for statistical machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP).*

Mihalcea, R. and Pedersen, T. (2003). An evaluation exercise for word alignment. In *Proceedings of the HLT-NAACL 2003 Workshop on Building and Using Parallel Texts: Data Driven Machine Translation and Beyond.*

Och, F. J. (1998). Ein beispielsbasierter und statistischer Ansatz zum maschinellen Lernen von natürlichsprachlicher Übersetzung. Master's thesis, Universität Erlangen-Nürnberg.

Och, F. J. (2002). *Statistical Machine Translation: From Single-Word Models to Alignment Templates.* PhD thesis, RWTH Aachen, Germany.

Och, F. J. (2003). Minimum error rate training for statistical machine translation. In *Proceedings of the 41st Annual Meeting of the Association of Computational Linguistics (ACL)*.

Och, F. J., Gildea, D., Sarkar, A., Khudanpur, S., Yamada, K., Fraser, A., Shen, L., Kumar, S., Smith, D., Jain, V., Eng, K., Jin, Z., and Radev, D. (2003). Syntax for machine translation. Technical report, John Hopkins University Summer Workshop `http://www.clsp.jhu.edu/ws2003/groups/translate/`.

Och, F. J. and Ney, H. (2002). Discriminative training and maximum entropy models for statistical machine translation. In *Proceedings of the 40th Annual Meeting of the Association of Computational Linguistics (ACL)*.

Och, F. J. and Ney, H. (2003). A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–52.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2001). BLEU: a method for automatic evaluation of machine translation. Technical Report RC22176(W0109-022), IBM Research Report.

Russel, S. and Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*. Prentice Hall, New Jersey.

Schafer, C. and Yarowski, D. (2003). Statistical machine translation using coercive two-level syntactic transduction. In *Proceedings of Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Stolke, A. (2002). Srilm - an extensible language modeling toolkit. In *Proceedings of the International Confonference on Spoken Language Processing*.

Tillmann, C. (2001). *Word Re-Ordering and Dynamic Programming based Search Algorithm for Statistical Machine Translation*. PhD thesis, RWTH Aachen, Germany.

Tillmann, C. (2003). A projection extension algorithm for statistical machine translation. In *Proceedings of Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Ueffing, N., Och, F. J., and Ney, H. (2002). Generation of word graphs in statistical machine translation. In *Proceedings of Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Venugopal, A., Vogel, S., and Waibel, A. (2003). Effective phrase translation extraction from alignment models. In *Proceedings of the 41st Annual Meeting of the Association of Computational Linguistics (ACL)*.

Vogel, S., Zhang, Y., Huang, F., Tribble, A., Venugopal, A., Zhao, B., and Waibel, A. (2003). The CMU statistical machine translation system. In *Proceedings of the Ninth Machine Translation Summit*.

Wu, D. (1997). Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational Linguistics*, 23(3).

Yamada, K. and Knight, K. (2001). A syntax-based statistical translation model. In *Proceedings of the 39th Annual Meeting of the Association of Computational Linguistics (ACL)*.

Zens, R., Och, F. J., and Ney, H. (2002). Phrase-based statistical machine translation. In *Proceedings of the German Conference on Artificial Intelligence (KI 2002)*.

Zhang, Y., Vogel, S., and Waibel, A. (2003). Integrated phrase segmentation and alignment algorithm for statistical machine translation. In *Proceedings of International Conference on Natural Language Processing and Knowledge Engineering (NLP-KE'03)*, Beijing, China.

Appendix: All Decoder Parameters

Parameters

`-f, -config` – specifies the location of the configuration file
`-t, -trace` – output is annotated with phrase translations used
`-v, -verbose` – allows for more detailed output: default (1), summary statistics (2), full progress report (3)
`-tm, -weight-t` – weight of the phrase translation table
`-lm, -weight-l` – weight of the language model
`-d, -weight-d` – weight of the distortion (reordering) model
`-w, -weight-w` – weight of the word penalty
`-ttable-limit` – number of phrase translations used for each foreign phrase (default 20)
`-ttable-threshold` – minimum probability for a phrase translation entry (default 0)
`-s, -stack` – maximum size of the beam, i.e., the hypothesis stacks (default 100)
`-b, -beam-threshold` – minimum value of a hypothesis relative to the best hypothesis in a stack (default 0.00001)
`-distortion-limit` – maximum distance between two input phrases that are translated to two neighboring output phrases (default 0, no limit)
`-bypass-marked` – allow to bypass the translations specified with the XML markup
`-weight-marked` – weight the specified translations (default 1) relative to model translations
`-l, -lattice` – creates word lattice files in a specified location
`-r, -rescore` – rescore a given translation with the model components
`-rd, -rescoredir` – rescore multiple files