Chapter 6
Decoding

In the two previous chapters we presented two models for machine translation, one based on the translation of words, and another based on the translation of phrases as atomic units. Both models were defined as mathematical formulae that, given a possible translation, assign a probabilistic score to it.

The task of **decoding** in machine translation is to find the best scoring translation according to these formulae. This is a hard problem, since there is an exponential number of choices, given a specific input sentence. In fact, it has been shown that the decoding problem for the presented machine translation models is NP-complete [Knight, 1999a]. In other words, exhaustively examining all possible translations, scoring them, and picking the best is computationally too expensive for an input sentence of even modest length.

decoding

In this chapter, we will present a number of techniques that make it possible to efficiently carry out the search for the best translation. These methods are called **heuristic search** methods. This means that there is no guarantee that they will find the best translation, but we do hope to find it often enough, or at least a translation that is very close to it.

heuristic search

Will decoding find a good translation for a given input? Note that there are two types of error that may prevent this. A **search error** is the failure to find the best translation according to the model, in other words, the highest-probability translation. It is a consequence of the heuristic nature of the decoding method, which is unable to explore the entire **search space** (the set of possible translations). It is the goal of this chapter to present methods that lead to a low search error.

search error

search space

model error

The other type of error is the **model error**. The highest-probability translation according to the model may not be a good translation at all. In this chapter, we are not concerned with this type of error. Of course, at the end of the day, what we want from our machine translation system is to come up with good translations that capture the meaning of the original and express it in fluent English. However, we have to address model errors by coming up with better models, such as adding components to a log-linear model as described in Section 5.3.1 on page 136. We cannot rely on **fortuitous search errors** that come up with a better translation even though it has lower probability than the highest-probability translation according to the model.

fortuitous search error

6.1 Translation Process

translation process

To gain some insight into the **translation process**, let us walk through one example of how a human translator might approach the translation of one sentence. We will do this with the phrase-based translation model in mind. It may be a bit too audacious to claim that the phrase-based model is a good model for human translation, but it serves our purpose here.

6.1.1 Translating a Sentence

In our example here, we are confronted with the following German sentence that we would like to translate into English:

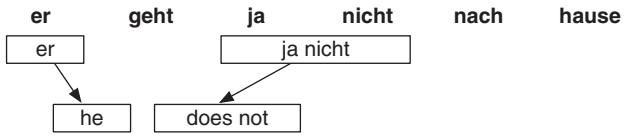
er geht ja nicht nach hause

With the phrase-based model in mind, we will try to translate this sentence by picking out short text chunks that we know how to translate. By doing this repeatedly we will able to stitch together an English translation. There are many ways to approach this: we will try to build the English sentence piece by piece, starting at the beginning.

Without further ado, we pick a phrase in the input and come up with a translation for it. Let us start modestly by taking the German word *er* and translating it as *he*:

In the illustration above we indicate the phrases we used by boxes, as we did in the chapter on phrase-based models. The boxes also help us to indicate which parts of the foreign input sentence we have already covered.

Note that we started with the first German word, but we are not required to do so. While we are building the English sentence in sequence, we may pick German words out of sequence. In fact, we will do so in the next step, where we translate *ja nicht* as *does not*:

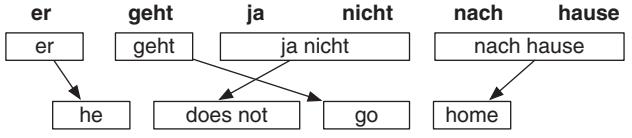


We have discussed the notion of **reordering**: input words and their translations do not have to occur in the same order. In this example, when translating from German to English, the negation particle (*nicht* or *not*) occurs in different positions: before the verb in English, after the verb in German.

reordering

In the translation process, this difference is accommodated by the possibility of picking words out of sequence in the input language when we are building the output sentence in sequence.

To finish up, we need to translate the remaining German words. We do so in two steps, first by translating *geht* as *go*, and then translating the phrase *nach hause* as *home*:



After these steps, we have exhausted all German words in the input sentence. Since we are not allowed to translate a word twice, we know that we are done.

Note that this example followed the phrase-based model, but it should be clear that the translation process for word-based models is similar. We also construct the output sentence from left to right in sequence and mark off input words.

6.1.2 Computing the Sentence Translation Probability

In the previous two chapters on word-based and phrase-based models for statistical machine translation, we introduced formulae to compute the probability of a translation given an input sentence.

Recall the basic formula for the phrase-based model (Equations 5.1 and 5.2):

$$\mathbf{e}_{\text{best}} = \text{argmax}_{\mathbf{e}} \prod_{i=1}^{I} \phi(\bar{f}_i|\bar{e}_i)\, d(\text{start}_i - \text{end}_{i-1} - 1)\, p_{\text{LM}}(\mathbf{e}) \qquad (6.1)$$

Several components contribute to the overall score: the phrase translation probability ϕ, the reordering model d and the language model p_{LM}. Given all $i = 1, ..., I$ input phrases $\bar{f}_i$ and output phrases $\bar{e}_i$ and their positions start_i and end_i, it is straightforward to compute the probability of the translation.

partial scoring

Moreover, during the translation process, we are able to compute **partial scores** for the partial translations we have constructed. So, instead of computing the translation probability for the whole translation only when it is completed, we can incrementally add to the score as we are constructing it.

Each time we add one phrase translation, we have to factor in the scores from the three model components in the formula:

- **Translation model**: When we pick an input phrase $\bar{f}_i$ to be translated as an output phrase $\bar{e}_i$, we consult the phrase translation table ϕ to look up the translation probability for this phrase pair.
- **Reordering model**: For reordering, we have to keep track of the end position in the input of the previous phrase we have translated, end_{i-1}. We also know where we are picking our current phrase and hence we have its start position in the input, start_i. Armed with these two numbers, we can consult the distortion probability distribution d to get the reordering cost.
- **Language model**: We will discuss language modeling in detail in Chapter 7. There we will describe how the probability of a sentence is computed using n-grams, for instance bigrams (2-grams): The probability that a sentence starts with *he* is $p_{\text{LM}}(\text{he}|\ <\text{s}>)$. The probability that *he* is followed by *does* is $p_{\text{LM}}(\text{does}|\text{he})$, and so on. This means, for our application of language models to translation, that once we add new words to the end of the sequence, we can compute their language model impact by consulting the language model p_{LM}. For this we need to keep track of the last $n - 1$ words we have added, since they form the history of the language model (the part of the probability we are conditioning on).

Reviewing the three model components – phrase translation, reordering, and language model – indicates that whenever we add a new translated phrase to our partially constructed translation, we can already compute its model cost. All we need to know is the identity of the phrase pair, its location in the input, and some information about the sequence we have constructed so far.

6.2 Beam Search

Having established two principles of the translation process – constructing the output sentence in sequence from left to right and incremental computation of sentence translation probability – we are now well prepared to face the reality of decoding.

6.2.1 Translation Options

First of all, in our illustration of the translation process in the previous section, we picked the phrase translations that made sense to us. The computer has less intuition. It is confronted with many options when deciding how to translate the sentence. Consulting the phrase translation table shows that many sequences in the input sentence could be translated in many different ways.

For a given input sentence, such as *er geht ja nicht nach hause*, we can consult our phrase table and look up all **translation options**, i.e., the phrase translations that apply to this input sentence. What does that look like? In Figure 6.1 we display the top four choices for all phrases in the input, using an actual phrase table (learnt from the Europarl corpus).

translation option

We can find the phrases for the right translation in this collection of translation options, but there are also many other choices. In fact, the actual Europarl phrase translation table provides a staggering 2,727 translation options for this sentence. The decoding problem is to find adequate phrase translations and place them together in a fluent English sentence. This is the search problem we are trying to solve.

As a side-note, the figure also illustrates nicely that word-by-word translation using the top-1 translation of each word is not a good approach to machine translation. For this sentence, it would result in the puzzling sequence of words *he is yes not after house*, which is not only bad English, but also a poor reflection of the meaning of the input sentence.

6.2.2 Decoding by Hypothesis Expansion

Armed with the notion of a translation process that builds the output sentence sequentially and a set of translation options to choose from, we can now appreciate the computer's decoding challenge.

In our search heuristic, we also want to build the output sentence from left to right in sequence by adding one phrase translation at a time.

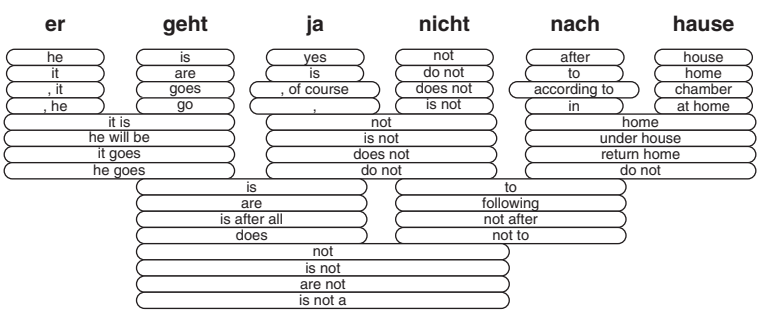


Figure 6.1 Translation options: Top four translations for phrases in the German input sentence *er geht ja nicht nach hause*. The phrases are taken from a phrase table learnt from the Europarl corpus. This is just the tip of the iceberg: the phrase table provides 2,727 translation options for this sentence.

hypothesis

During the search, we are constructing partial translations that we call **hypotheses**.

From a programming point of view, you can think of a hypothesis as a data structure that contains certain information about the partial translation, e.g., what English output words have been produced so far, which foreign input words have been covered, the partial score, etc.

empty hypothesis

The starting point is the **empty hypothesis**. The empty hypothesis has no words translated and hence empty output. Since no probabilities have been factored in at this point, its partial probability score is 1.

hypothesis expansion

We are **expanding** a hypothesis when we pick one of the translation options and construct a new hypothesis.

For instance, initially, we may decide to translate the German one-word phrase *er* as its first option *he*. This means placing the English phrase *he* at the beginning of the sentence, checking off the German word *er*, and computing all the probability costs, as described in the previous section. This results in a new hypothesis that is linked to the initial empty hypothesis.

But, we may instead decide to expand the initial hypothesis by translating the German word *geht* as *are* (the second translation option for this one-word phrase – see Figure 6.1). Again, we have to mark off the German, attach the English, and add the translation costs. This results in a different hypothesis.

The decoding process of hypothesis expansion is carried out recursively, as illustrated in Figure 6.2. Hypotheses are expanded into new

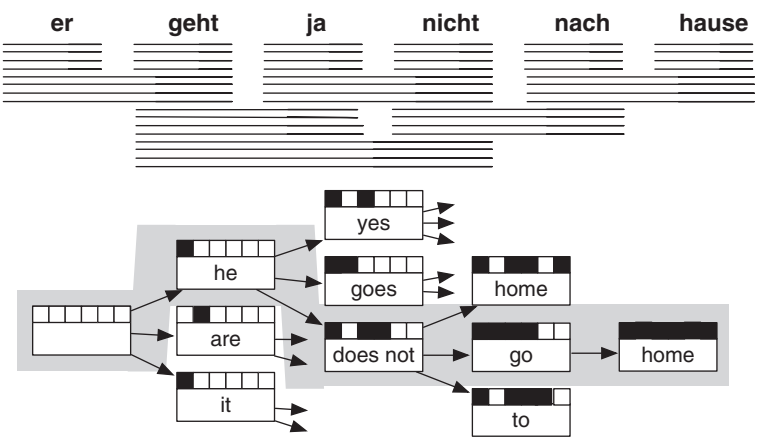


Figure 6.2 Decoding process: Starting with the initial empty hypothesis, hypotheses are expanded by picking translation options (indicated by the lines; see Figure 6.1 for details) and applying the current hypothesis. This results in a new hypothesis in the search graph. Each hypothesis in this illustration is pictured as a box containing the most recently added English, a coverage vector of translated German words (the squares on top, filled black if covered), and a pointer from its parent hypothesis.

hypotheses by applying phrase translations to input phrases that have not been covered yet. This process continues until all hypotheses have been expanded.

A hypothesis that covers all input words cannot be expanded any further and forms an end point in the **search graph**. Given all such completed hypotheses, we have to find the one with the best probability score: this is the end point of the best path through the search graph. In other words, when we track back (using the pointers) through the search graph, we find the best-scoring translation.

search graph

6.2.3 Computational Complexity

Time for reflection: We have proposed a search heuristic that lets us find all possible translations for the input sentence, given a set of translation options. This heuristic is fairly efficient: if two translations differ only in the way the last word is translated, they share the same hypothesis path up to the last hypothesis, and this common path has to be computed only once.

However, the size of the search space grows roughly exponentially in the length of the input sentence. This makes our search heuristic computationally prohibitive for any large sentence. Recall that machine translation decoding has been shown to be NP-complete.

If we want to be able to translate long sentences, we need to address this complexity problem. We will do so in two steps. In the next section we show that we can reduce the number of hypotheses by hypothesis recombination. However, this is not sufficient, and we have to resort to pruning methods that make estimates of which hypotheses to drop early, at the risk of failing to find the best path.

6.2.4 Hypothesis Recombination

Hypothesis recombination takes advantage of the fact that matching hypotheses can be reached by different paths. See Figure 6.3 for an illustration. Given the translation options, we may translate the first two German words separately into *it* and *is*. But it is also possible to get

Hypothesis recombination

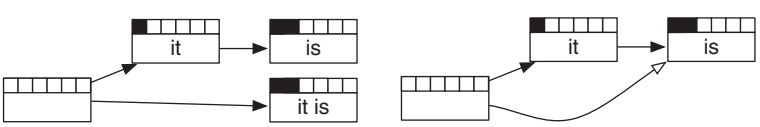


Figure 6.3 Recombination example: Two hypothesis paths lead to two matching hypotheses: they have the same number of foreign words translated, and the same English words in the output, but different scores. In the heuristic search they can be recombined, and the worse hypothesis is dropped.

this translation with a single translation option that translates the two German words as a phrase into *it is*.

Note that the two resulting hypotheses are not identical: although they have identical coverage of already translated German words and identical output of English words, they do differ in their probability scores. Using two phrase translations results in a different score than using one phrase translation.

One of the hypotheses will have a lower score than the other. The argument that we can drop the worse-scoring hypothesis is as follows: The worse hypothesis can never be part of the best overall path. In any path that includes the worse hypothesis, we can replace it with the better hypothesis, and get a better score. Thus, we can drop the worse hypothesis.

Identical English output is not required for hypothesis recombination, as the example in Figure 6.4 shows. Here we consider two paths that start with different translations for the first German word *er*, namely *it* and *he*. But then each path continues with the same translation option, skipping one German word, and translating *ja nicht* as *does not*.

We have two hypotheses that look similar: they are identical in their German coverage, but differ slightly in the English output. However, recall the argument that allows us to recombine hypotheses. If any path starting from the worse hypothesis can also be used to extend the better hypothesis with the *same* costs, then we can safely drop the worse hypothesis, since it can never be part of the best path.

It does not matter in our two examples that the two hypothesis paths differ in their first English word. All subsequent phrase translation costs are independent of already generated English words. The language model is sensitive to only a few of the latest words in the output. A typical trigram language model uses as its history the two latest English words. However, in our example, these two words are identical: both hypothesis paths have output that ends in *does not*. In conclusion, from a subsequent search point of view the two hypotheses are identical and can be recombined.

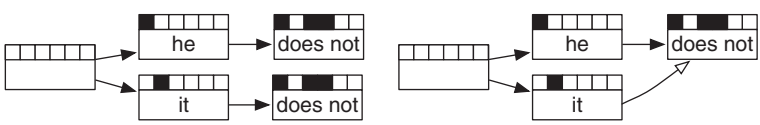


Figure 6.4 More complex recombination example: In heuristic search with a trigram language model, two hypotheses can be recombined. The hypotheses are the same with respect to subsequent costs. The language model considers the last two words produced (*does not*), but not the third-to-last word. All other later costs are independent of the words produced so far.

You may have noted that in the illustrations, we did not simply erase the worse hypothesis, but kept an arc that connects its parent hypothesis with the better hypothesis. For finding the best translation, this arc is not required, but we will later discuss methods to also find the second best path, the third best path, and so on. For this purpose, we would like to preserve the full search graph. Keeping the arcs enables this.

Note that different model components place different constraints on the possibility of hypothesis recombination:

- **Translation model:** The translation models we discussed here treat each phrase translation independent of each other, so they place no constraints on hypothesis recombination.
- **Language model:** An n-gram language model uses the last $n - 1$ words as history to compute the probability of the next word. Hence, two hypotheses that differ in their last $n - 1$ words cannot be recombined.
- **Reordering model:** We introduced two reordering models in the previous chapter on phrase-based models. For both models, the sentence position of the last foreign input phrase that was translated matters in addition to the sentence position of the current foreign input phrase. So, if two hypotheses differ with respect to this sentence position, they cannot be recombined.

Recombining hypotheses is very helpful in reducing spurious ambiguity in the search. The method reduces the problem of having to consider two hypothesis paths that differ only in internal representation such as different phrase segmentation. This leads to a tighter and more efficient search. However, it does not solve the problem that machine translation decoding has exponential complexity. To address this problem, we have to employ riskier methods for search space reduction, which we will discuss next.

6.2.5 Stack Decoding

Given that the decoding problem is too complex to allow an efficient algorithm that is guaranteed to find the best translation according to the model, we have to resort to a heuristic search that reduces the search space. If it appears early on that a hypothesis is bad, we may want to drop it and ignore all future expansions. However, we can never be certain that this hypothesis will not redeem itself later and lead to the best overall translation.

Dropping hypotheses early leads to the problem of search error. In heuristic search, search error is a fact of life that we try to reduce as much as possible, but will never eliminate. To know with certainty that a hypothesis is bad, we need to find its best path to completion. However, this is too expensive.

How can we make good guesses at which hypotheses are likely to be too bad to lead to the best translation? The exponential explosion of the search space creates too many hypotheses, and at some point fairly early on we need to take a set of hypotheses, compare them, and drop the bad ones. It does not reduce the search space enough only to choose among hypotheses that are *identical* from a search point of view (this is what we already do with hypothesis recombination); however, we want to consider hypotheses that are at least comparable.

hypothesis stack
pruning

To this end, we would like to organize hypotheses into **hypothesis stacks**. If the stacks get too large, we **prune** out the worst hypotheses in the stack. One way to organize hypothesis stacks is based on the **number of foreign words translated**. One stack contains all hypotheses that have translated one foreign word, another stack contains all hypotheses that have translated two foreign words in their path, and so on.

Note the following concern. Are hypotheses that have the same number of words translated truly comparable? Some parts of the sentence may be easier to translate than others, and this should be taken into consideration. We will save this thought for later, and come back to it in Section 6.3 on future cost estimation.

Figure 6.5 illustrates the organization of hypotheses based on number of foreign words covered. Stack 0 contains only one hypothesis: the empty hypothesis. Through hypothesis expansion (applying a translation option to a hypothesis) additional words are translated, and the resulting new hypothesis is placed in a stack further down.

The notion of organizing hypotheses in stacks and expanding hypotheses from stacks leads to a very compact search heuristic. Figure 6.6 gives a pseudo-code specification. We iterate through all stacks, all hypotheses in each stack, and all translation options to create new hypotheses.

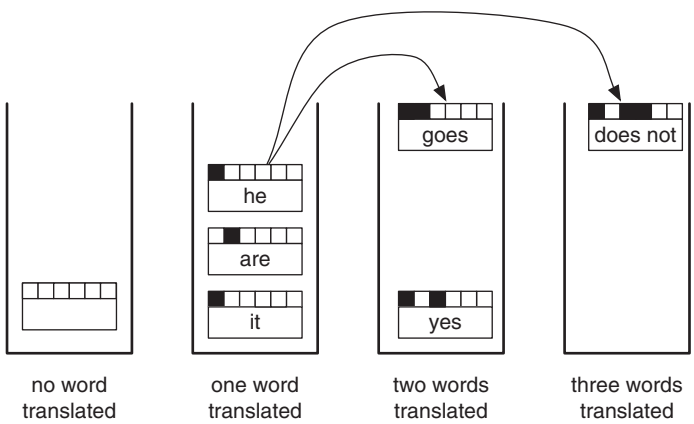


Figure 6.5 Hypothesis stacks: All hypotheses with the same number of words translated are placed in the same stack (coverage is indicated by black squares at the top of each hypothesis box). This figure also illustrates hypothesis expansion in a stack decoder: a translation option is applied to a hypothesis, creating a new hypothesis that is dropped into a stack further down.

```
 1: place empty hypothesis into stack 0
 2: for all stacks 0...n−1 do
 3:   for all hypotheses in stack do
 4:     for all translation options do
 5:       if applicable then
 6:          create new hypothesis
 7:          place in stack
 8:          recombine with existing hypothesis if possible
 9:          prune stack if too big
10:       end if
11:     end for
12:   end for
13: end for
```

Figure 6.6 Pseudo-code for the stack decoding heuristic described in Section 6.2.5.

6.2.6 Histogram Pruning and Threshold Pruning

Of course, the exponential nature of machine translation decoding leads to very large numbers of hypotheses in the stacks. Since each stack contains comparable hypotheses (they cover the same number of foreign input words), we use pruning to drop bad hypotheses based on their partial score.

There are two approaches to pruning. In **histogram pruning**, we keep a maximum number n of hypotheses in the stack. The stack size n has a direct relation to decoding speed. Under the assumption that all stacks are filled and all translation options are applicable all the time, the number of hypothesis expansions is equal to the maximum stack size times the number of translation options times the length of the input sentence.

histogram pruning

Let us put the computational time complexity of decoding into a formula:

$$O(\text{max stack size} \times \text{translation options} \times \text{sentence length}) \quad (6.2)$$

Note that the number of translation options is linear with sentence length. Hence, the complexity formula can be simplified to

$$O(\text{max stack size} \times \text{sentence length}^2) \quad (6.3)$$

If hypothesis expansion is the only computational cost, the cost of stack decoding is quadratic with sentence length, quite an improvement from the original exponential cost. This improvement comes, however, at the risk of not finding the best translation according to the model.

The second approach to pruning, **threshold pruning**, exploits the following observation: sometimes there is a large difference in score between the best hypothesis in the stack and the worst, sometimes they are very close. Histogram pruning is very inconsistent with regard to

threshold pruning

pruning out bad hypotheses: sometimes relatively bad hypotheses are allowed to survive, sometimes hypotheses that are only slightly worse than the best hypothesis are pruned out.

Threshold pruning proposes a fixed threshold α, by which a hypothesis is allowed to be worse than the best one in the stack. If the score of a hypothesis is α times worse than the best hypothesis, it is pruned out.

This threshold can be visualized as a beam of light that shines through the search space. The beam follows the (presumably) best hypothesis path, but with a certain width it also illuminates neighboring hypotheses that differ not too much in score from the best one. Hence the name **beam search**, which is the title of this section.

beam search

The impact on the computational cost of decoding, given different thresholds α, is less predictable, but roughly the same quadratic complexity holds here too. In practice, today's machine translation decoders use both histogram pruning and threshold pruning. Threshold pruning has some nicer theoretical properties, while histogram pruning is more reliable in terms of computational cost. By the choice of the two limits, stack size n and beam threshold α, it is easy to emphasize one pruning method over the other.

6.2.7 Reordering Limits

If you are familiar with the statistical approach to speech recognition or tagging methods for problems such as part-of-speech tagging and syntactic chunking, the decoding approach we have presented so far will sound vaguely familiar (in all these problems, input is mapped to output in a sequential word-by-word process). However, one distinct property makes machine translation decoding much harder: the input may be processed not linearly, but in any order.

Reordering is one of the hard problems in machine translation, both from a computational and from a modeling point of view. For some language pairs, such as French–English, limited local reordering almost suffices (nouns and adjectives flip positions). Translating other languages into English requires major surgery. In German, the main verb is often at the end of the sentence. The same is true for Japanese, where almost everything is arranged in the opposite order to English: the verb is at the end, prepositions become post-positioned markers, and so on.

The phrase-based machine translation model we presented in the previous chapter, and for which we are now devising decoding methods, is fairly weak in regard to large-scale restructuring of sentences. We proposed some advanced models only for local reordering. Large-scale restructuring is driven by syntactic differences in language, and we will

need to come back to that problem in the later chapters on syntax-based approaches to machine translation.

Nevertheless, popular language pairs among statistical machine translation researchers, such as French–English, Chinese–English, and Arabic–English, appear to be a good fit for phrase-based models despite their limited capabilities in global reordering. Some early work in statistical machine translation even ignored completely the reordering problem and limited itself to so-called **monotone decoding**, where the input sentence has to be processed in order.

monotone decoding

A compromise between allowing any reordering and allowing only monotone decoding is achieved by the introduction of **reordering limits**. When taking phrases out of sequence, a maximum of d words may be skipped.

reordering limit

In practice, statistical machine translation researchers found that beyond a certain reordering limit d (typically 5–8 words), machine translation performance does not improve and may even deteriorate. This deterioration reflects either more search errors or a weak reordering model that does not properly discount bad large-scale reordering.

Note the effect of the introduction of reordering limits on computational complexity: with limited reordering, only a limited number of translation options is available at any step. In other words, the number of translation options available no longer grows with sentence length. This removes one of the linear dependencies on sentence length and reduces Equation (6.3) to

$$O(\text{max stack size} \times \text{sentence length}) \tag{6.4}$$

With decoding speed linear in sentence length, we are now very comfortable translating even the longest sentences. Pruning limits (in the form of maximum stack size and beam threshold) allow us to trade off between translation speed and search error.

6.3 Future Cost Estimation

We have argued for a beam-search stack decoding algorithm, where we organize comparable hypotheses (partial translations) into stacks and prune out bad hypotheses when the stacks get too big. Pruning out hypotheses is risky, and a key to minimizing search errors is to base the pruning decision on the right measure.

6.3.1 Varying Translation Difficulty

We proposed comparing all hypotheses that have the same number of foreign words translated and pruning out the ones that have the worst probability score. However, there is a problem with this approach. Some

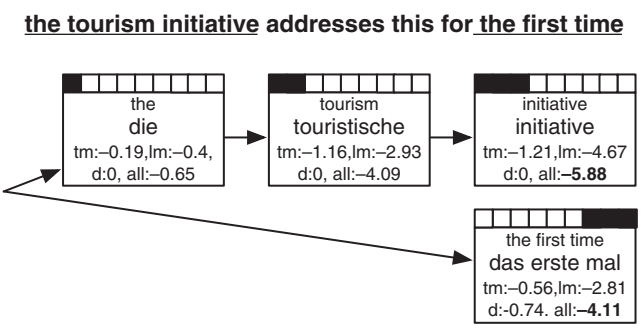


Figure 6.7 Translating the easy part first: The hypothesis that translates (out of order) *the first time* has a better score (−4.11) than the correct hypothesis that starts with the hard words *the tourism initiative* at the beginning of the sentence. Both translated three words, and the worse-scoring (−5.88) but correct hypothesis may be pruned out. Scores are scaled log-probabilities from an English–German model trained on Europarl data (tm: translation model, lm: language model, d: reordering model).

parts of the sentence may be easier to translate than others, and hypotheses that translate the easy parts first are unfairly preferred to ones that do not.

For example, the translation of unusual nouns and names is usually more expensive than the translation of common function words. While translation model costs are similar, the language model prefers common words over unusual ones. See Figure 6.7 for an example of this problem. When translating the English sentence *the tourism initiative addresses this for the first time*, the hard words are at the beginning, and the easy words towards the end. A hypothesis that skips ahead and translates the easy part first has a better probability score than one that takes on the hard part directly, even taking reordering costs into account. In the example, translating first *the first time* yields a better score (−4.11) than translating first *the tourism initiative* (−5.88). Pruning based on these scores puts the latter hypothesis at an unfair disadvantage. After all, the first hypothesis still has to tackle the hard part of the sentence.

future cost

We would like to base pruning decisions not only on the probability score of the hypothesis, but also on some measure of **future cost**, i.e., the expected cost of translating the rest of the sentence. Keep in mind that it is computationally too expensive to compute the expected cost exactly, because this would involve finding the best completion of the hypothesis, which is precisely the search problem we are trying to solve.

outside cost
rest cost

Hence, we need to estimate the future cost. This is also called **outside cost** or **rest cost** estimation. Adding a future cost estimate to the partial probability score leads to a much better basis for pruning decisions.

6.3.2 Estimating Future Cost for Translation Options

Computing a future cost estimate means estimating how hard it is to translate the untranslated part of the input sentence. How can we efficiently estimate the expected translation cost for part of a sentence? Let us start simply and consider the estimation of the cost of applying a specific translation option. Recall that there are three major model components, and each affects the model score:

- **Translation model:** For a given translation option, we can quickly look up its translation cost from the phrase translation table.
- **Language model:** We cannot compute the actual language model cost of applying a translation option when we do not know the preceding words. A good estimate in most cases, however, is the language model cost without context: the unigram probability of the first word of the output phrase, the bigram probability of the second word, and so on.
- **Reordering model:** We know very little about the reordering of the translation option, so we ignore this for the future cost estimation.

Once we have future cost estimates for all translation options, we can estimate the cheapest cost for translating any span of input words covered by the phrase translation table.

Why the cheapest? When translating the rest of the sentence, we want to find the best path to completion. Of course, we would like to use the cheapest translation option, unless a combination of other translation options gives an even better score. So, we do not expect to have a completion more expensive than the one given by the cheapest options (of course, there is no guarantee, because the language model cost is only an estimate and the real cost may turn out differently).

Figure 6.8 illustrates the future costs computed for all spans covered by the translation table in our example sentence. For some parts of the sentence, we can find only one-word matches in the translation table. For instance, the English *tourism initiative* does not occur in the training

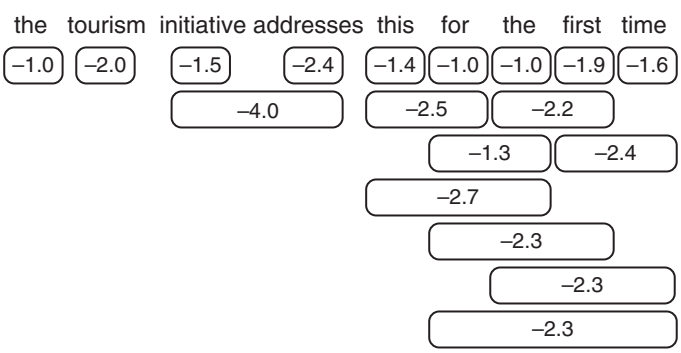


Figure 6.8 Future cost estimates of input phrases covered by the translation table: For each covered span, the cost of the cheapest translation option is displayed. Future cost estimates take translation model and language model probabilities into account.

data, so we do not have a phrase translation for it. Not surprisingly, we have a translation for the English *for the first time* in the translation table, since it is a commonly occurring phrase.

The values of the future cost estimates for the translation options confirm our intuition that the translation of common function words is cheaper than the translation of unusual nouns – translating *tourism* is twice as expensive (−2.0) as translating either *for* or *the* (−1.0). Using a phrase translation, e.g., for *for the first time* (−2.3), is generally cheaper than using one-word phrases (−1.0, −1.0, −1.9, and −1.6 add up to −5.5). The costs reported here are weighted log-probabilities from an English–German translation model trained on the Europarl corpus.

6.3.3 Estimating Future Cost for Any Input Span

We need future cost estimates not only for spans of input words that are covered by phrase translations in the translation table, but also for longer spans. We will compute these using the future cost estimates for the covered spans. There are complex interactions between translation options in actual search (e.g., language model effects and reordering), but we will ignore them for the purpose of future cost estimation.

We can very efficiently compute the future cost using a dynamic programming algorithm that is sketched out in Figure 6.9. The cheapest cost estimate for a span is either the cheapest cost for a translation option or the cheapest sum of costs for a pair of spans that cover it completely.

Figure 6.10 gives cost estimates for all spans in our example sentence. The first four words in the sentence are much harder to translate (*the tourism initiative addresses*, estimate: −6.9) than the last four words (*for the first time*, estimate: −2.3).

```
 1: for length = 1 .. n do
 2:   for start = 1...n+1-length do
 3:     end = start+length
 4:     cost(start,end) = infinity
 5:     cost(start,end) = translation option cost estimate if exists
 6:     for i=start..end-1 do
 7:       if cost(start,i) + cost(i+1,end) < cost(start,end) then
 8:         update cost(start,end)
 9:       end if
10:     end for
11:   end for
12: end for
```

Figure 6.9 Pseudo-code to estimate future costs for spans of any length.

first word	future cost estimate for n words (from first)								
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
the	–1.0	–3.0	–4.5	–6.9	–7.7	–8.7	–9.0	–10.0	–10.0
tourism	–2.0	–3.5	–5.9	–6.7	–7.7	–8.0	–9.0	–9.0	
initiative	–1.5	–3.9	–4.7	–5.7	–6.0	–6.9	–7.0		
addresses	–2.4	–3.1	–4.1	–4.5	–5.4	–5.5			
this	–1.4	–2.4	–2.7	–3.7	–3.7				
for	–1.0	–1.3	–2.3	–2.3					
the	–1.0	–2.2	–2.3						
first	–1.9	–2.4							
time	–1.6								

Figure 6.10 Future cost estimates indicate the difficulty of translating parts of the input sentence *the tourism initiative addresses this for the first time*. Numbers are scaled log-scores from an English–German translation model trained on the Europarl corpus. Some words are easier to translate than others, especially common functions words such as *for* (−1.0) and *the* (−1.0), as opposed to infrequent verbs or nouns such as *addresses* (−2.4) or *tourism* (−2.0). The four-word phrase *for the first time* (−2.3) is much easier to translate than the three-word phrase *tourism initiative addresses* (−5.9).

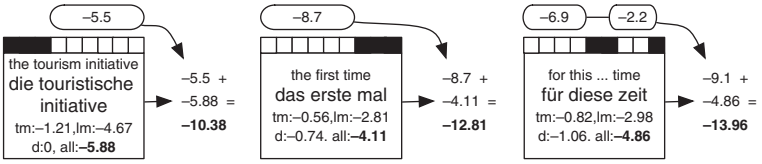


Figure 6.11 Combining probability score and future cost: While the probability score alone may be misleading, adding an estimate of the expected future cost gives a more realistic basis for pruning out bad hypotheses. Here, the hypothesis that tackles the hard part of the sentence *the tourism initiative* has the worse score (−5.88 against −4.11, −4.86), which is offset by a cheaper future cost estimate (−5.5 against −8.7, −9.1), resulting in an overall better score than the two competing hypotheses that cover simpler parts of the sentence (−10.38 against −12.81, −13.96).

6.3.4 Using Future Cost in the Search

We now have in our hands the tool we need to discount hypotheses that translate the easy part of the sentence first. By adding up the partial score and the future cost estimate, we have a much better measure of the quality of a hypothesis. Basing pruning decisions on this measure will lead to lower search error than using just the probability score.

Recall our example, shown again in Figure 6.11, where the hypothesis that skipped the start of the sentence and translated the easy *the first time* had a better score (−4.11) than the hypothesis that tackled head-on *the tourism initiative* (−5.88). Adding in the future cost estimates levels the playing field and puts the better hypothesis ahead (−10.38 vs. −12.81).

It may happen that skipping words leads to several contiguous spans in the input sentence that have not been covered yet. In this case, we simply add up the cost estimates for each span. Since we are ignoring interaction between translation options in the future cost estimation, we can also ignore interaction between uncovered spans that are separated by translated words.

Note that we have ignored reordering costs in the future cost estimates so far. However, we may want to take these into account. If a hypothesis creates coverage gaps, this means that some reordering cost will have to be added further down in the path. Computing the minimum distance of the required jumps gives a good measure of the cheapest expected reordering cost. Adding this to the future cost estimate may reduce search errors.

6.4 Other Decoding Algorithms

We presented in detail a beam-search stack decoder for phrase-based models, which is the most commonly used decoding algorithm. The same type of decoder may be used for word-based models. We now review several other decoding algorithms that have been proposed in the literature, to conclude this chapter.

6.4.1 Beam Search Based on Coverage Stacks

Organizing hypotheses in stacks based on the number of translated foreign input words introduced the additional complexity of future cost estimation. However, if we were to have a stack for each span of foreign input words covered, we could do away with that.

If we only compare hypotheses that translate the same span of foreign words, their future cost estimates, as we defined them, are the same, so we can ignore them. Note that it is still possible to make search errors: while one hypothesis may look better than the alternatives at a given point in the search, it may end with an English word that leads to worse language model scores in the next step, and will not be part of the best path.

coverage stacks

The problem with such **coverage stacks** is that there is an exponential number of them, which makes this approach computationally infeasible. However, recall our argument in Section 6.2.7 for the use of reordering limits. A reordering limit will reduce the number of possible foreign word coverage vectors to a number that is linear with sentence length (albeit still exponential with the reordering limit). So, coverage stack decoding with reordering limits is practical.

6.4.2 A* Search

The beam search we have presented here is very similar to **A* search**, which is described in many artificial intelligence textbooks. A* search allows pruning of the search space that is risk free, in other words, prevents search error. A* search

A* search puts constraints on the heuristic that is used to estimate future cost. A* search uses an **admissible heuristic**, which requires that the estimated cost is never an overestimate. Note how this can be used to safely prune out hypotheses: if the partial score plus estimated future cost for a hypothesis is worse than the score for the cheapest completed hypothesis path, we can safely remove it from the search. admissible heuristic

Admissible heuristic for machine translation decoding

The future cost heuristic that we described in detail in Section 6.3 is not an admissible heuristic: it may over- or underestimate actual translation costs. How can we adapt this heuristic? We ignore reordering costs and use the actual phrase translation cost from the translation table, so we do not run the risk of overestimating these components of cost.

However, the language model cost is a rough estimate, ignoring the preceding context, so it may over- or underestimate the actual translation cost. We can get optimistic language model estimates instead by considering the most advantageous history. For instance, for the probability of the first word in a phrase, we need to find the highest probability given any history.

Search algorithm

For A* search to be efficient, we need to quickly establish an early candidate for the cheapest actual completed cost. Hence, we follow a depth-first approach, illustrated in Figure 6.12.

We first expand a hypothesis path all the way to completion. To accomplish this, we perform all possible hypothesis expansions of a hypothesis and then proceed with the one that has the cheapest cost estimate, e.g., partial score and the heuristic future cost estimate. Then, we can explore alternative hypothesis expansions and discard hypotheses whose cost estimate is worse than the score for the cheapest completed hypothesis.

Depth-first search implies that we prefer the expansion of hypotheses that are closest to completion. Only when we have expanded all hypotheses that can be completed in one step, do we back off to hypotheses that can be completed in two steps, and so on. Contrast this with the **breadth-first search** in the stack decoding algorithm that we described earlier: here, we reach completed hypotheses only after Depth-first search

breadth-first search

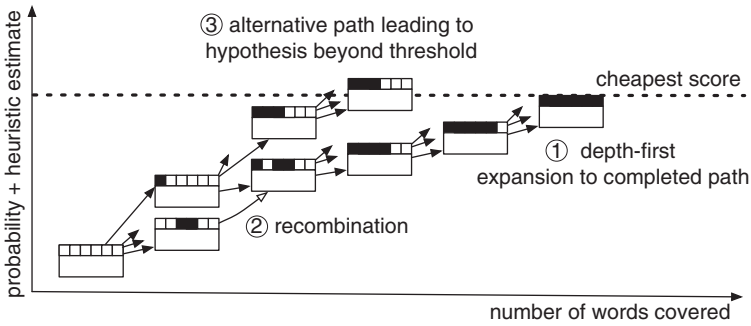


Figure 6.12 A* search: (1) First, one hypothesis path is expanded to completion, establishing the cheapest actual score. (2) Hypotheses may also be recombined as in stack decoding. (3) If an alternative expansion leads to a cost estimate better than the cheapest score threshold, it is pruned. Not pictured: New cheaper hypothesis tightens the cheapest score threshold. Note that hypothesis expansion never worsens the overall cost estimate, since actual costs are never better than estimated costs.

covering the whole breadth of the extensions that cover one foreign word, then two foreign words, and so on.

agenda-driven search

A third search strategy, called **agenda-driven search**, is to always expand the cheapest hypothesis. We organize hypotheses that have not been expanded yet on a prioritized list, i.e., the agenda. By following this agenda, we expect to find more quickly better completed hypotheses that can improve the cheapest actual cost. Once no hypothesis with a better cost estimate than the cheapest score exists, we are done.

While A* search is very efficient in cutting down the search space, there is no guarantee that it finishes in polynomial time (recall that machine translation decoding is NP-complete). Another problem with A* search is that it requires an admissible heuristic, meaning a future cost estimate that is never an underestimate. This may lead to less realistic cost estimates.

6.4.3 Greedy Hill-Climbing Decoding

greedy hill climbing

A completely different approach to decoding is **greedy hill climbing**. First, we generate a rough initial translation, and then we apply a number of changes to improve it. We do this iteratively, until no improving step can be found.

The initial translation may be as simple as the lexical translation of every word, without reordering, with the best translation for each word. We may consider language model probabilities when picking the word translations, or perform monotone decoding using the full translation table.

The steps to improve a translation may include the following:

- change the translation of a word or phrase;
- combine the translation of two words into a phrase;
- split up the translation of a phrase into two smaller phrase translations;
- move parts of the English output into a different position;
- swap parts of the English output with the English output at a different part of the sentence.

All possible steps are considered, and only those that lead to an improvement are applied. There are a number of variants. We may always apply a step, if it leads to an improvement, or search for the best improvement at any point. We may also want to look two steps ahead to be able to apply two steps that first decrease translation probability, but then more than make up for it.

The advantage of this decoding approach is that we always have a full translation of the sentence in front of us. This enables the addition of model components that score **global properties**. For instance, does the English output contain a verb? Also, it is an **anytime method**: at any time, we can stop the translation process if we are satisfied with the output or are bound by time constraints (e.g., a maximum of 5 seconds of decoding time per sentence).

global properties

anytime method

The main disadvantage of this decoding method is its limitation to a much smaller search space than the beam-search approach. We may get stuck in local optima, where a sequence of two or more steps is needed to reach a better translation. In contrast, the dynamic programming element of recombining hypotheses allows the beam search to efficiently consider many more possible translations than the hill-climbing search.

6.4.4 Finite State Transducer Decoding

Finally, a popular choice in building machine translation decoders is the use of finite state machine toolkits. The search graph of the machine translation decoding process is in essence a huge probabilistic **finite state transducer**. Transitions between states are hypothesis expansions. Transition probability is the added model costs incurred by that expansion.

finite state transducer

Using finite state machine toolkits has great appeal. We do not need to implement a decoding algorithm. We only need to construct the finite state transducer for the translation of one sentence. This implies defining the state space and the transitions between states using the phrase translation table.

On the other hand, we are not able to integrate heuristics for future cost estimation, but have to rely on the general-purpose search of

the finite state toolkit. As a consequence, researchers typically restrict reordering fairly severely, because otherwise the number of states in the search graphs becomes unmanageably big.

A purpose-built decoder for machine translation is usually more efficient. However, if the goal is to quickly try many different models, finite state transducers provide faster turnaround.

6.5 Summary

6.5.1 Core Concepts

This chapter described **decoding** algorithms that use statistical machine translation models and find the best possible translation for a given input sentence. Due to the exponential complexity of the search space, we employ **heuristic search** methods.

Heuristic search is not guaranteed to find the best translation and hence may lead to **search errors**. Contrast this type of error to **model errors** which are the result of the model giving a bad translation the highest probability. While it is possible to have **fortuitous search errors**, which occur when the search finds a translation that is lower scoring (according to the model) but better (according to human assessment), we cannot rely on these.

We described the **translation process** of building a translation from an input and used it as motivation for the search algorithm. Of course, in statistical machine translation we have to deal with many **translation options** given an input sentence. Search is formulated as a succession of **hypotheses** (in essence partial translations), starting with an **empty hypothesis** (nothing is translated), and using **hypothesis expansion** to build new ones.

The search space can be reduced by **hypothesis recombination**, but this is not sufficient. We hence proposed a **stack decoding** heuristic, in which hypotheses are organized in **hypothesis stacks**, based on the number of foreign words translated. The size of the stacks is reduced by **pruning**. We distinguished between **histogram pruning**, which limits the number of hypotheses per stack, and **threshold pruning**, which discards hypotheses that are worse than the best hypothesis in a stack by at least a certain factor. The search space is typically also reduced by imposing **reordering limits**.

For a fair comparison of hypotheses that have covered different parts of the input sentence, we have to take into account an estimate of the **future cost** of translating the rest of the input sentence. This estimate is also called **rest cost** or **outside cost**.

We reviewed a number of alternative search heuristics. Beam search based on **coverage stacks** is a viable alternative when strict reordering limits are employed, and it does away with future cost estimation. **A* search** is guaranteed to find the best translation when the future cost estimate is an **admissible heuristic**.

A* search is usually performed as **depth-first search** (complete a translation as soon as possible), while the stack decoding is an example of **breadth-first search** (expand all hypotheses at the same pace). Another search strategy uses an **agenda** to expand the hypotheses that are most promising at any given time.

Greedy hill climbing starts with an initial translation, and recursively improves it by changing it in steps. This search method exhibits **anytime** behavior, meaning it can be interrupted at any time (e.g., by a time constraint) and have ready a complete translation (maybe not the best possible).

Finally, **finite state transducers** have been used for machine translation decoding, which takes advantage of existing finite state toolkits.

6.5.2 Further Reading

Stack decoding – The stack decoding algorithm presented here has its roots in work by Tillmann *et al.* [1997], who describe a dynamic programming algorithm, similar to techniques for hidden Markov models, for monotone decoding of word-based models. Allowing for reordering makes decoding more complex. Efficient A* search makes the translation of short sentences possible [Och *et al.*, 2001]. However, longer sentences require pruning [Wang and Waibel, 1997; Nießen *et al.*, 1998] or restrictions on reordering [Tillmann and Ney, 2000] or both [Tillmann and Ney, 2003]. Ortiz-Martínez *et al.* [2006] compare different types of stack decoding with varying numbers of stacks. Delaney *et al.* [2006] speed up stack decoding with A* pruning. Moore and Quirk [2007a] stress the importance of threshold pruning and the avoidance of expanding doomed hypotheses. Some efficiency may be gained by collapsing contexts that are invariant to the language model [Li and Khudanpur, 2008].

Reordering constraints – Matusov *et al.* [2005] constrain reordering when the input word sequence was consistently translated monotonically in the training data. Zens and Ney [2003], Zens *et al.* [2004] and Kanthak *et al.* [2005] compare different reordering constraints and their effect on translation performance, including the formal grammar ITG constraint, which may be further restricted by insisting on a match to source-side syntax [Yamamoto *et al.*, 2008]. Similarly, reordering

may be restricted to maintain syntactic cohesion [Cherry, 2008]. Ge *et al.* [2008] integrate other linguistically inspired reordering models into a phrase-based decoder. Dreyer *et al.* [2007] compare reordering constraints in terms of oracle BLEU, i.e., the maximum possible BLEU score.

Decoding for word-based models – Several decoding methods for word-based models are compared by Germann *et al.* [2001], who introduce a greedy search [Germann, 2003] and integer programming search method. Search errors of the greedy decoder may be reduced by a better initialization, for instance using an example-based machine translation system for seeding the search [Paul *et al.*, 2004]. A decoding algorithm based on alternately optimizing alignment (given translation) and translation (given alignment) is proposed by Udupa *et al.* [2004].

Decoding with finite state toolkits – Instead of devising a dedicated decoding algorithm for statistical machine translation, finite state tools may be used, both for word-based [Bangalore and Riccardi, 2000, 2001; Tsukada and Nagata, 2004; Casacuberta and Vidal, 2004], alignment template [Kumar and Byrne, 2003], and phrase-based models. The use of finite state toolkits also allows for the training of word-based and phrase-based models. The implementation by Deng and Byrne [2005] is available as the MTTK toolkit [Deng and Byrne, 2006]. Similarly, the IBM models may be implemented using graphical model toolkits [Filali and Bilmes, 2007]. Pérez *et al.* [2007] compare finite state implementations of word-based and phrase-based models.

Decoding complexity – Knight [1999a] showed that the decoding problem is NP-complete. Udupa and Maji [2006] provide further complexity analysis for training and decoding with IBM models.

6.5.3 Exercises

1. (⋆) Given the following input and phrase translation options:

das	*ist*	*das*	*Haus*	*von*	*Nikolaus*
the	*is*	*the*	*house*	*of*	*Nicholas*
that		*that*		*from*	
that's		*house*		*Nicholas'*	

decode the input by hand with the stack decoding that we described in Figure 6.6 on page 165. For simplification, assume no reordering. Draw the search graph constructed during decoding assuming recombination when using

(a) a trigram language model;
(b) a bigram language model;
(c) a unigram language model.

2. (⋆) Given the input and phrase translation options:

das	*ist*	*das*	*Haus*
the	*is*	*the*	*house*
that	*'s*	*that*	*home*
this	*are*	*this*	*building*

   how many possible translations for the input sentence exist
   (a) without reordering;
   (b) with reordering?
3. (⋆⋆) Implement a stack decoder, as sketched out in pseudo-code in Figure 6.6 on page 165. For simplification, you may ignore language model scoring. Test the decoder on the example phrase tables from Questions 1 and 2, after adding arbitrary translation probabilities,
   (a) without recombination;
   (b) with recombination under a trigram language model.
4. (⋆⋆) Install Moses[1] and follow the step-by-step guide to train a model using the Europarl corpus.[2]
5. (⋆⋆⋆) Recent research supports the use of maximum entropy classifiers to include more context in translation decisions. Using the open-source Moses system as a starting point, implement maximum entropy classifiers that predict
   (a) an output phrase given an input phrase and other words in the sentence as features;
   (b) reordering distance based on word movement, and syntactic properties over the input part-of-speech tags and input syntactic tree.
6. (⋆⋆⋆) Implement A* search in Moses.

[1] Available at `http://www.statmt.org/moses/`
[2] Available at `http://www.statmt.org/europarl/`