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:

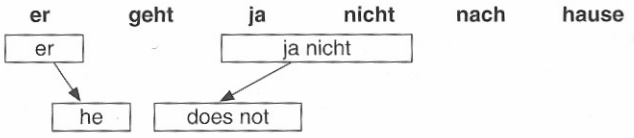
er geht ja nicht nach hause

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.

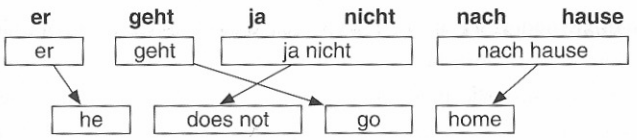
Note that we started with the first German word, but we are not required to do so. While we are building the English sentence in sequence, we may pick German words out of sequence. In fact, we will do so in the next step, where we translate *ja nicht* as *does not*:



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.

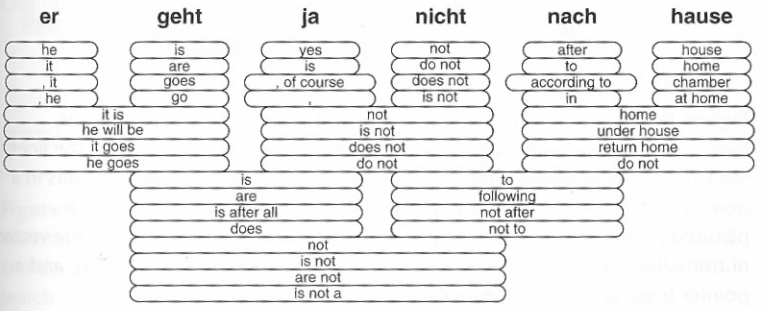


Figure 6.1 Translation options: Top four translations for phrases in the German input sentence *er geht ja nicht nach hause*. The phrases are taken from a phrase table learnt from the Europarl corpus. This is just the tip of the iceberg: the phrase table provides 2,727 translation options for this sentence.

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

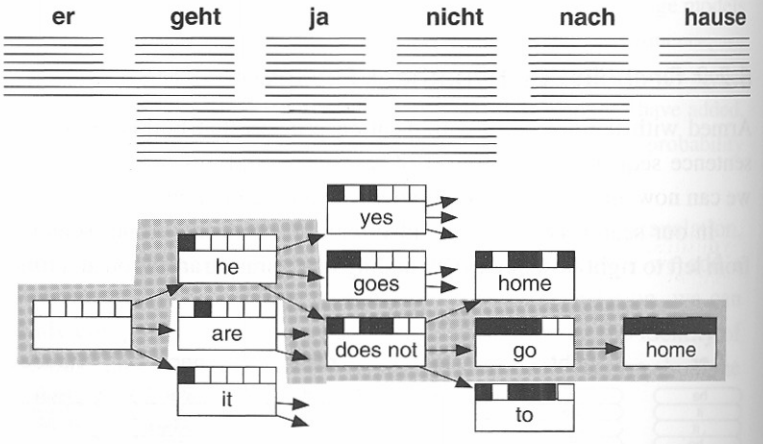


Figure 6.2 Decoding process: Starting with the initial empty hypothesis, hypotheses are expanded by picking translation options (indicated by the lines; see Figure 6.1 for details) and applying the current hypothesis. This results in a new hypothesis in the search graph. Each hypothesis in this illustration 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

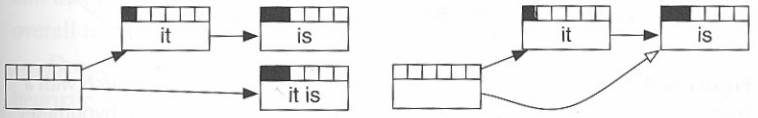


Figure 6.3 Recombination example: Two hypothesis paths lead to two matching hypotheses: they have the same number of foreign words translated, and the same English words in the output, but different scores. In the heuristic search they can be recombined, and the worse hypothesis is dropped.

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.

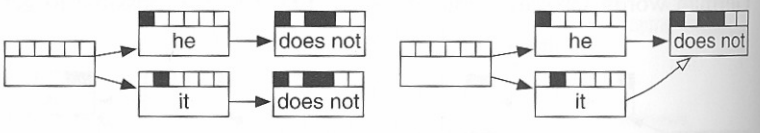


Figure 6.4 More complex recombination example: In heuristic search with a trigram language model, two hypotheses can be recombined. The hypotheses are the same with respect to subsequent costs. The language model considers the last two words produced (*does not*), but not the third-to-last word. All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

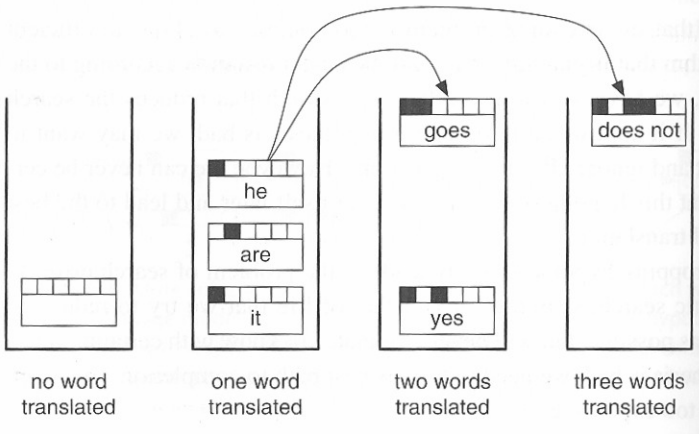


Figure 6.5 Hypothesis stacks: All hypotheses with the same number of words translated are placed in the same stack (coverage is indicated by black squares at the top of each hypothesis box). This figure also illustrates hypothesis expansion in a stack decoder: a translation option is applied to a hypothesis,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