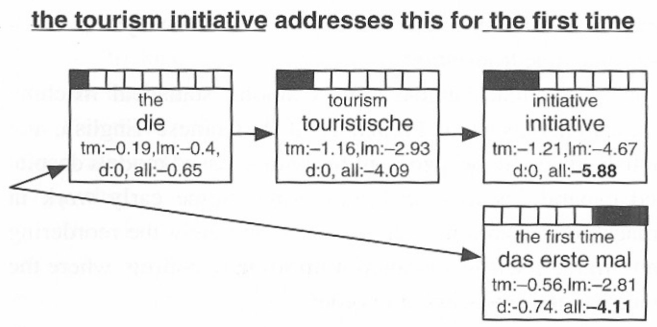

Figure 6.7 Translating the easy part first: The hypothesis that translates (out of order) *the first time* has a better score (−4.11) than the correct hypothesis that starts with the hard words *the tourism initiative* at the beginning of the sentence. Both translated three words, and the worse-scoring (−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

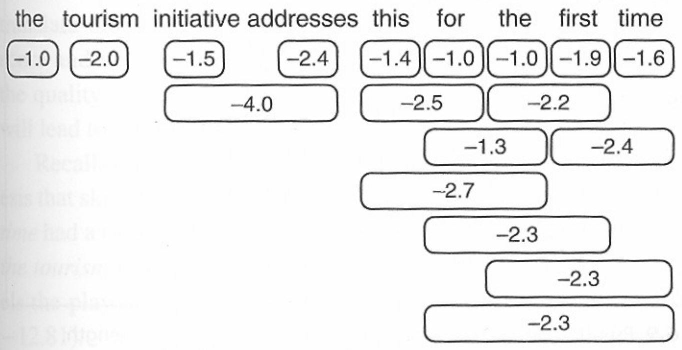

Figure 6.8 Future cost estimates of input phrases covered by the translation table: For each covered span, the cost of the cheapest translation option is displayed. Future cost estimates take translation model and language model probabilities into account.

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).

```
for length = 1 .. n do
  for start = 1...n+1-length do
     end = start+length
     cost(start,end) = infinity
     cost(start,end) = translation option cost estimate if exists
     for i=start..end-1 do
       if cost(start,i) + cost(i+1,end) < cost(start,end) then
          update cost(start,end)
       end if
     end for
  end for
end for
```

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

first	future cost estimate for n words (from first)								
word	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).

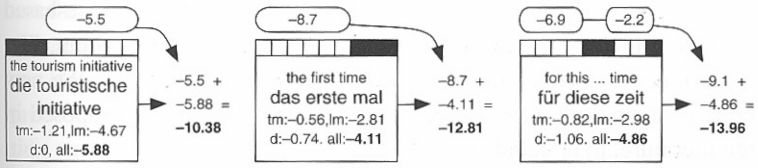

Figure 6.11 Combining probability score and future cost: While the probability score alone may be misleading, adding an estimate of the expected future cost gives a more realistic basis for pruning out bad hypotheses. Here, the hypothesis that tackles the hard part of the sentence *the tourism initiative* 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

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.

admissible heuristic

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 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

breadth-first search

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

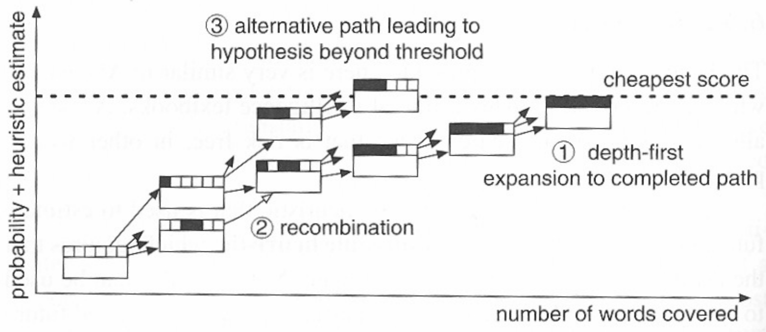
③ alternative path leading to hypothesis beyond threshold
cheapest score
① depth-first expansion to completed path
② recombination
probability + heuristic estimate
number of words covered

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/`