

# Multistack Decoding in Statistical Machine Translation

Michael E. Jahr<sup>1</sup>  
Honors Thesis  
Symbolic Systems Program  
Stanford University

June 2001

<sup>1</sup>[jahr@cs.stanford.edu](mailto:jahr@cs.stanford.edu)

## **Abstract**

In a machine translation system, decoding is the process of finding the most likely translation according to previously learned parameters. The success of any such system is highly dependent on the quality of its decoder. Stack decoders were the first decoders developed for statistical machine translation, and they represent a good compromise between search thoroughness and efficiency. In this thesis I review the models used in IBM's Candide system, and then describe a multistack decoder developed for use with the models. I also compare the performance of the multistack decoder to that of two other decoding algorithms.

## Approval

I deem this thesis completely satisfactory  
in scope and quality for award of Honors in  
Symbolic Systems at Stanford University.

---

Christopher Manning  
Assistant Professor  
Computer Science and Linguistics  
Stanford University

---

Kevin Knight  
Research Assistant Professor  
Computer Science  
University of Southern California

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Outline . . . . .	2
1.2	Language Choices . . . . .	2
1.3	Acknowledgments . . . . .	2
<b>I</b>	<b>Statistical Machine Translation</b>	<b>4</b>
<b>2</b>	<b>Probabilistic Background</b>	<b>4</b>
2.1	The Noisy Channel Model . . . . .	4
2.2	Bayesian Division of Labor . . . . .	4
<b>3</b>	<b>Language Modeling</b>	<b>6</b>
<b>4</b>	<b>Translation Modeling</b>	<b>8</b>
4.1	Notation . . . . .	8
4.2	IBM's Lower Models . . . . .	9
4.2.1	Model 1: Word Translation . . . . .	10
4.2.2	Model 2: Local Alignment . . . . .	11
4.3	IBM's Higher Models . . . . .	11
4.3.1	Model 3: Fertility . . . . .	13
4.3.2	Model 4: Class-Based Alignment . . . . .	15
4.3.3	Model 5: Non-Deficient Alignment . . . . .	17
4.4	Training . . . . .	18
<b>II</b>	<b>Decoding</b>	<b>19</b>
<b>5</b>	<b>The Decoding Problem</b>	<b>19</b>
5.1	Existing Decoders . . . . .	19
5.1.1	Stack Decoders . . . . .	19
5.1.2	Other Decoders . . . . .	20
<b>6</b>	<b>Stack-Based Decoding</b>	<b>21</b>

<b>7</b>	<b>Operations</b>	<b>23</b>
7.1	Notation . . . . .	24
7.2	Open and Closed Hypotheses . . . . .	24
7.3	Operation Definitions . . . . .	25
7.3.1	Add-closed . . . . .	26
7.3.2	Add-open . . . . .	26
7.3.3	Add-zfert . . . . .	27
7.3.4	Add-null . . . . .	28
7.3.5	Extend-open . . . . .	28
7.3.6	Extend-closed . . . . .	31
7.4	Sample Decoding . . . . .	31
<b>8</b>	<b>Optimizations</b>	<b>31</b>
8.1	Operation Complexity . . . . .	31
8.1.1	Zero-Fertility Operations . . . . .	33
8.2	Redundant Hypotheses . . . . .	35
8.3	Hypothesis Data Structure . . . . .	36
<b>9</b>	<b>Results</b>	<b>38</b>
<b>10</b>	<b>Future Work</b>	<b>42</b>
<b>11</b>	<b>Conclusion</b>	<b>43</b>

## List of Tables

1	Fields of a hypothesis. . . . .	24
2	Assignments made by $\text{ADD-CLOSED}(e, j)$ . . . . .	26
3	Assignments made by $\text{ADD-OPEN}(e, j)$ . . . . .	27
4	Assignments made by $\text{ADD-ZFERT}(e)$ . . . . .	27
5	Assignments made by $\text{ADD-NULL}(j)$ . . . . .	28
6	Assignments made by $\text{EXTEND-OPEN}(j)$ . . . . .	29
7	Definition of the $\text{DPROB}$ function. . . . .	29
8	Assignments made by $\text{EXTEND-CLOSED}(j)$ . . . . .	31
9	Top zero-fertility words in the Hansards. . . . .	34
10	An efficient hypothesis representation. . . . .	37
11	Comparison of decoders on sets of 101 test sentences, using a bigram language model. . . . .	40
12	Comparison of decoders on sets of 101 test sentences, using a trigram language model. . . . .	41

## List of Figures

1	An alignment of French and English strings. . . . .	9
2	Relationships between decoder input and output for speech recognition and machine translation. . . . .	22
3	Three cases for distortion probabilities after an $\text{EXTEND}$ op- eration. . . . .	30
4	A sample decoding. . . . .	32

# 1 Introduction

The foundation of machine translation is often dated to the 1949 publication of a memorandum by Warren Weaver, a prominent mathematician of the time.<sup>1</sup> The memorandum contained a letter he wrote in 1947 to his friend Norbert Wiener, wherein he wrote:

Recognising fully, even though necessarily vaguely, the semantic difficulties because of multiple meanings, etc., I have wondered if it were unthinkable to design a computer which would translate. Even if it would translate only scientific material (where the semantic difficulties are very notably less), and even if it did produce an inelegant (but intelligible) result, it would seem to me worth while.

Also knowing nothing official about, but having guessed and inferred considerable about, powerful new mechanized methods in cryptography—methods which I believe succeed even when one does not know what language has been coded—one naturally wonders if the problem of translation could conceivably be treated as a problem in cryptography. When I look at an article in Russian, I say ‘This is really written in English, but it has been coded in some strange symbols. I will now proceed to decode.’ [19]

Progress in machine translation at first seemed very promising, with many researchers in the 1950s predicting fully automatic systems within a few decades. After a few initial successes, however, subsequent progress was disappointing, and the publication of the ALPAC report, which in 1966 recommended that funding for machine translation be diverted elsewhere, was a devastating blow to the young field. The seemingly insurmountable obstacle discovered in this decade is that accurate translation seems impossible without a perfect understanding of the text to be translated. After the ALPAC report, funding quickly dried up all over the world, and research in the field was sharply reduced over the next two decades. In the 1980s, there

---

<sup>1</sup>There had been earlier proposals for various kinds of mechanized translating machines, but these were for the most part entirely independent of the first computer machine translation research in the early 1950s.

was a resurgence of interest in machine translation, encouraged by advances in the size and speed of computers and advancements in computer science in general. Much of this new research was along the same lines as in the 1950s: carefully crafting elaborate grammars, dictionaries, and translation rules by hand.

A group at IBM's Thomas J. Watson Research Center, however, suspected that this approach was untenable in the long run. Instead, they proposed to let a system use machine learning techniques to learn the translation process on its own, by examining a large set of documents translated in multiple languages. The Candide machine translation system, developed in the early 1990s, was based on exactly this premise.

## **1.1 Outline**

The underpinnings of statistical machine translation, as well as the mathematical models of the Candide system, are described in detail in Part I. The workings of the stack decoder, as well as experimental results, are described in Part II.

## **1.2 Language Choices**

The Candide system requires a large set of translated documents to train its models. One particularly attractive source of such documents is the Canadian parliamentary proceedings, known as "Hansards." Because they used the Hansards, the goal of the IBM team was to translate French to English, and they used these languages in all their papers. In the present work we will follow this convention, and address the problem of translating from French to English. It should be clear that one could easily substitute, with varying degrees of success, any other pair of languages here, given a large enough set of translated documents.

## **1.3 Acknowledgments**

This work really began all the way back in 1999, at the summer workshop of the Center for Language and Speech Processing of Johns Hopkins University.

I owe a debt of gratitude to Fred Jelinek, head of the CLSP and organizer of the workshop, both for selecting me to participate in it, and also for taking a chance by awarding two undergraduates a grant for continuing research begun there.

The Statistical Machine Translation team at the 1999 workshop was headed by Kevin Knight, who was one of my two advisors for this thesis. It was Kevin who first introduced me to the field of natural language processing, and under his direction that I conducted almost all my research. Nearly everything I know about machine translation I owe to Kevin's exceptional ability to deconstruct the most complex ideas in simple but precise terms. I again worked with Kevin over the next summer, 2000, this time at the University of Southern California's Information Sciences Institute, where I also received extensive help from other members of the REWRITE group, especially Daniel Marcu and Uli Germann.

Finally, I actually began writing in the first half of 2001, during which time I relied on the capable guidance and editorial assistance of Chris Manning, my Stanford advisor. In particular, I'd like to thank Chris for his extraordinary patience in putting up with what has possibly been the most drawn-out thesis process in Stanford history.

This thesis is based on work supported in part by the National Science Foundation under Grant No. IIS-9820687.

## Part I

# Statistical Machine Translation

## 2 Probabilistic Background

### 2.1 The Noisy Channel Model

We wish to model the process of translating a string of French words  $f$  into a string of English words  $e$ . To do so, we create a very simple, extremely unrealistic model of how it is that French is produced in the first place. This model must be very simple, because we are going to try to teach a computer how to use it. Specifically, we assume that French speakers initially have in mind an English string, denoted  $e$ , which they process mentally to produce the French string, denoted  $f$ . Put another way, we might say that when French speakers try to express the English string  $e$ , the output is somehow garbled, and it comes out as the French string  $f$ . In communication theory, this is known as a noisy channel model: there is a signal  $e$  which passes through a noisy channel and is corrupted into  $f$ . We must estimate the original input  $e$  to the noisy channel, given only the observable output  $f$ . In other words, we model the process of translation as the task of finding a reconstruction  $\hat{e}$  of the English string  $e$  that the French speaker used to produce a given French string  $f$ .

### 2.2 Bayesian Division of Labor

It is almost always the case that a string of French words can be translated into English in several different ways; that is, there are several  $e$ 's which could be acceptable translations of a given  $f$ . As some of these  $e$ 's will be better translations than others, we can imagine assigning each one a number based on how good a translation it is of  $f$ . Let us call this number  $\Pr(e|f)$ ; we can conceptualize its value as the probability that  $f$  be chosen as a translation of  $e$ .

Probabilistically, we are trying to find the  $\hat{e}$  that maximizes  $\Pr(\hat{e}|f)$ .

By Bayes' theorem, we can write

$$\Pr(\hat{e} | \mathbf{f}) = \frac{\Pr(\hat{e}) \Pr(\mathbf{f} | \hat{e})}{\Pr(\mathbf{f})}. \quad (1)$$

Realizing that the denominator is independent of  $\hat{e}$ , we find that maximizing  $\Pr(\hat{e} | \mathbf{f})$  is the same as maximizing the product  $\Pr(e) \Pr(\mathbf{f} | e)$ . In this way, we arrive at what [5] call the “Fundamental Equation of Statistical Machine Translation”:

$$\hat{e} = \underset{e}{\operatorname{argmax}} \Pr(e) \Pr(\mathbf{f} | e) \quad (2)$$

Upon first glance, Equation 2 might seem to be a roundabout way of estimating the quantity in which we're interested. Instead of calculating both  $\Pr(e)$  and  $\Pr(\mathbf{f} | e)$ , why not just calculate  $\Pr(e | \mathbf{f})$  directly? The answer is that by estimating two numbers instead of one, we are making the task easier by dividing it into two subproblems: the *language modeling problem* and the *translation modeling problem*. The language model evaluates whether  $e$  is fluent English, without taking  $\mathbf{f}$  into account at all. The translation model evaluates whether  $\mathbf{f}$  is a good translation of  $e$ . Because of this division of labor, the translation model need only evaluate whether  $e$  has the right words in roughly the right places, letting the language model ensure fluency.

By way of illustrating this division of labor, let us examine a French string  $\mathbf{f} =$  “la téléphone sonne” and three possible English translations:

**“the dog barks”** This is fluent English, so the language model will give it a high score. However, it has very little to do with  $\mathbf{f}$ , so the translation model will give it low marks.

**“telephone rings the”** This case is just the opposite: it is not fluent English, but seems to have the right words to be a good translation of  $\mathbf{f}$ . The translation model might give it a high score, but the language model will not.

**“the telephone rings”** Finally, we have a case which is both fluent English and a good translation of  $\mathbf{f}$ ; this string will be assigned a high

score by both models.

In this way, the language model and translation can individually give high scores to poor translations, but when used together, they produce much better results. Were we to calculate  $\Pr(\mathbf{e} | \mathbf{f})$  directly, we would have to develop a model capable of performing both these tasks at once, which would be more difficult.

### 3 Language Modeling

The task of the language model is to assign a probability  $\Pr(\mathbf{e})$  to an English string  $\mathbf{e} = e_1, e_2, \dots, e_m$ , where  $e_i$  is the  $i$ th word in  $\mathbf{e}$ .<sup>2</sup> Without loss of generality, we can write

$$\begin{aligned}\Pr(\mathbf{e}) &= \Pr(e_1, e_2, \dots, e_m) \\ &= \Pr(e_1) \Pr(e_2 | e_1) \dots \Pr(e_m | e_1, e_2, \dots, e_{m-1}).\end{aligned}\tag{3}$$

In doing so, we reduce the language modeling problem to the task of calculating the probability of an English word given all the words that precede it in the string. The preceding words are called the *history*. In this formulation, our model would require a prohibitively large number of parameters to account for all possible sentences, since the last few words of the string depend on many other words. To reduce the number of parameters, we need to find a way to combine like histories. One way to do this is to make a *Markov assumption* that each word depends only on a few recent words rather than the entire history. An *n-gram* model makes exactly such an assumption, where all histories that end in the same  $n - 1$  words are treated as equivalent. The most common values used are  $n = 2$  or  $3$ , respectively known as *bigram* and *trigram* models. By convention, the words bigram and trigram refer to a sequence of two or three consecutive words, respectively.

Among the appealing features of n-gram models is that they are very easy to train and use. Broadly speaking, training a trigram model involves

---

<sup>2</sup>We use italics ( $e$ ) to refer to individual words, and bold ( $\mathbf{e}$ ) to refer to strings of words.

taking a large volume of text and counting the number of times each trigram occurs. When we're done counting, we normalize by the total number of trigrams we've seen, and the result is a large probability table. To use the model, we look up the probability of each trigram in a sentence and multiply them together.

For example, consider the sentence "The dog barks." By convention, we add sentence boundary markers, denoted  $*$ , at the beginning and end of the sentence, so we are trying to estimate  $\Pr(*, the, dog, barks, *)$ . We can break this down according to Equation 3:

$$\Pr(\mathbf{e}) = \Pr(*) \Pr(the | *) \Pr(dog | *, the) \Pr(barks | *, the, dog) \times \Pr(* | *, the, dog, barks) \quad (4)$$

We know that  $\Pr(*) = 1$  since every sentence starts with this symbol by definition. Under the bigram model, we can simplify Equation 4 as follows:

$$\Pr(\mathbf{e}) = \Pr(the | *) \Pr(dog | the) \Pr(barks | dog) \Pr(* | barks) \quad (5)$$

This is a very broad sketch of n-gram models; many variations and improvements have been made on this basic theme. One topic that merits particular attention here is the treatment of previously unseen n-grams. Even with the limited history employed in trigrams, the vast majority of possible trigrams will not occur in even the largest training corpora. There are many schemes for dealing with this; common ones are *smoothing* which assigns some low probability to unseen instances, and *backing off*, which employs bigram and unigram probabilities to fill the gaps in the trigram model.

Language modeling is a problem which has been very well studied; indeed, one of the advantages of separating the language and translation models is that it allows us to apply results from other areas of statistical language processing. Furthermore, it is straightforward to experiment with different combinations of translation and language models to find the pair that works best. A statistical parser, for instance, could easily take the place of the

n-gram model described here. Although they are somewhat naïve from a linguistics perspective, n-grams have proven remarkably useful in many settings, notably in the domain of speech recognition. In fact, it is surprisingly difficult for even much more complex models to improve on the predictive power of n-grams.

For a thorough discussion of n-gram models as well as other types of language models, the reader is referred to [13].

## 4 Translation Modeling

IBM’s Candide machine translation system employs five translation models, each of which builds on the previous ones. The five models are called, somewhat predictably, Models 1 through 5. Each successive model provides an increasingly sophisticated method of estimating  $\Pr(\mathbf{f}, \mathbf{a} | \mathbf{e})$ . We will examine each of IBM’s models in turn.

### 4.1 Notation

Before we dive into the details of IBM’s translation models, it is worthwhile to introduce some notation that we will use throughout this paper. We are already well acquainted with  $\mathbf{e}$  and  $\mathbf{f}$ , the English and French strings. We define  $l$  to be the length of  $\mathbf{e}$ , and  $m$  to be the length of  $\mathbf{f}$ , and use  $i$  and  $j$  to refer to positions in  $\mathbf{e}$  and  $\mathbf{f}$ , where  $i = 0, 1, 2, \dots, l$  and  $j = 1, 2, \dots, m$ .<sup>3</sup> The variables  $e$  and  $f$  will represent individual English and French words, so that  $e_i$  and  $f_j$  refer to the  $i$ th word of  $\mathbf{e}$  and the  $j$ th word of  $\mathbf{f}$ . Finally, a variable written with both subscript and superscript, as in  $f_1^m$  will refer to  $f_1, f_2, \dots, f_m$ .

One concept that will be useful is the *alignment*, which we denote  $\mathbf{a}$ . An alignment indicates which words in  $\mathbf{e}$  are translations of which words in  $\mathbf{f}$ , as shown in Figure 1. The idea is to connect two words if they express the same concept in their respective languages; if two words are connected,

---

<sup>3</sup>By convention, the special position  $i = 0$  is reserved for the null word  $e_0$ , described later in this section. Indices in  $\mathbf{e}$  and  $\mathbf{f}$  are numbered from 1.

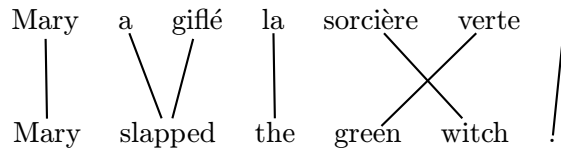


Figure 1: An alignment of French and English strings.

we say they are aligned. Note that alignments need not be one-to-one: it is often the case that a single word in one string will translate to several words in the other string, or that a word not have a direct translation at all. In theory, an alignment  $\mathbf{a}$  can consist of any set of connections. IBM's models, however, are restricted to alignments where each French word is connected to at most one English word. Because of this restriction, we can represent an alignment as a series of English indices  $a_1, a_2, \dots, a_m$ . In this representation, word 1 of  $\mathbf{f}$  is connected to word  $a_1$  of  $\mathbf{e}$ . In other words,  $f_1$  is connected to  $e_{a_1}$ ,  $f_2$  is connected to  $e_{a_2}$ , and so on. If a French word  $f_j$  is not connected to any words in  $\mathbf{e}$ , we say it is connected to the *null word*, denoted  $e_0$ , and assign  $a_j$  the value 0.

## 4.2 IBM's Lower Models

To make the translation modeling problem somewhat more tractable, IBM's models actually estimate the quantity  $\Pr(\mathbf{f}, \mathbf{a} | \mathbf{e})$ , or the probability of an French string and a specific alignment given an English string.

To calculate  $\Pr(\mathbf{f}, \mathbf{a} | \mathbf{e})$  for a specific pair of strings, it is to our advantage to break it down into more specific quantities. Without loss of generality, we can write

$$\Pr(\mathbf{f}, \mathbf{a} | \mathbf{e}) = \Pr(m | \mathbf{e}) \Pr(\mathbf{a} | m, \mathbf{e}) \Pr(\mathbf{f} | \mathbf{a}, m, \mathbf{e}) \quad (6)$$

$$= \Pr(m | \mathbf{e}) \prod_{j=1}^m \Pr(a_j | a_1^{j-1}, m, \mathbf{e}) \times \prod_{j=1}^m \Pr(f_j | a_1^m, f_1^{j-1}, m, \mathbf{e}). \quad (7)$$

Note that this is not an approximation; we have simply rewritten a joint

probability as the product of several conditional probabilities. One way of explaining Equation 7 is to say that it describes a process by which we generate a French string  $\mathbf{f}$  and an alignment  $\mathbf{a}$  given an English string  $\mathbf{e}$ . According to this process, we first choose the length of  $\mathbf{f}$ ,  $m$ , based on what we know about  $\mathbf{e}$ . We then iterate over each of the positions in  $\mathbf{f}$  and align each one with a position in  $\mathbf{e}$ . We align each position in  $\mathbf{f}$  given on our knowledge of  $\mathbf{e}$ ,  $m$ , and the alignments we have made so far. (Note that we align the positions in  $\mathbf{f}$  before we choose the words to fill them!) We then iterate over the positions in  $\mathbf{f}$  again, and fill each one with a word. This choice is based on  $\mathbf{e}$ ,  $m$ , the alignments of all positions in  $\mathbf{f}$ , and the words we've chosen for  $\mathbf{f}$  up to this point.

IBM's Models 1 and 2 are based on Equation 7.

#### 4.2.1 Model 1: Word Translation

We start with the idea that every French word  $f$  is possibly a translation of every English word  $e$ , with some probability. We capture this intuition by defining a *word translation probability*, denoted  $t(f | e)$ , as the probability that  $e$  is translated as  $f$ . Furthermore, we make the unrealistic independence assumption that that this word translation probability does not depend on the context of either word. In terms of Equation 7, we define

$$t(f_j | e_{a_j}) \equiv \Pr(f_j | a_1^j, f_1^{j-1}, m, \mathbf{e}). \quad (8)$$

Model 1 assumes every alignment is equally probable, so we assign them a uniform value:  $\Pr(a_j | a_1^{j-1}, f_1^{j-1}, m, \mathbf{e}) = (l + 1)^{-1}$ . The remaining term in Equation 7 is  $\Pr(m | \mathbf{e})$ ; we assume it is independent of both  $m$  and  $\mathbf{e}$ . We define  $\Pr(m | \mathbf{e}) \equiv \epsilon$ , where  $\epsilon$  is some small, fixed constant. Making the above substitutions to Equation 7, we arrive at the probability of an alignment and translation under Model 1:

$$\Pr(\mathbf{f}, \mathbf{a} | \mathbf{e}) = \frac{\epsilon}{(l + 1)^m} \prod_{j=1}^m t(f_j | e_{a_j}). \quad (9)$$

### 4.2.2 Model 2: Local Alignment

Model 2 adds parameters to estimate the probability of alignments as well as translations. Specifically, we introduce *alignment probabilities*, denoted  $a(i | j, m, l)$ , as the probability that the  $j$ th word in a French string of length  $m$  be aligned with the  $i$ th word in an English string of length  $l$ . Recall that in Equation 7 we assign  $a_j$  before  $f_j$ ; for this reason alignment probabilities cannot depend on the French words they are aligning. As with the word translation probabilities in Model 1, we assume that alignment probabilities are independent of their context, so we define

$$a(a_j | j, m, l) \equiv \Pr(a_j | a_1^{j-1}, f_1^{j-1}, m, \mathbf{e}). \quad (10)$$

We can now update Equation 9 to include these alignment probabilities:

$$\Pr(\mathbf{f}, \mathbf{a} | \mathbf{e}) = \epsilon \prod_{j=1}^m t(f_j | e_{a_j}) a(a_j | j, m, l). \quad (11)$$

### 4.3 IBM's Higher Models

While Models 1 and 2 are based on the formulation in Equation 7, Models 3, 4, and 5 break down  $\Pr(\mathbf{f}, \mathbf{a} | \mathbf{e})$  in a different way. Central to this new formulation is the idea of *fertility*, defined as the number of French words with which an English word is aligned. We use  $\phi$  to denote a set of fertilities;  $\phi_i$  is the fertility of the  $i$ th word of  $\mathbf{e}$ ,  $e_i$ . Because a French word  $f_j$  can be aligned with the null word  $e_0$ , we also have  $\phi_0$ , called the *null fertility*. In this way, we should be able to account for all the words in  $\mathbf{f}$ , so that  $\sum_{i=0}^l \phi_i = m$ . We define a *tablet* to be the list of French words aligned with an English word. We use  $\tau$  to refer to a set of tablets, where the tablet of  $e_i$  is  $\tau_i$  and the  $k$ th French word of  $\tau_i$  is  $\tau_{ik}$ . Note that there are  $\phi_i$  words in the tablet  $\tau_i$ . We define a *permutation* to be an assignment of the words in a tablet to positions in  $\mathbf{f}$ . We use  $\pi$  to refer to a permutation;  $\pi_i$  is the permutation of the  $i$ th tablet  $\tau_i$ , and  $\pi_{ik}$  is the position in  $\mathbf{f}$  of the  $k$ th word of  $\tau_i$ .

Using this notation, we can now provide a replacement for Equation 7:

$$\begin{aligned}
\Pr(\boldsymbol{\tau}, \boldsymbol{\pi} | \mathbf{e}) &= \Pr(\boldsymbol{\phi} | \mathbf{e}) \Pr(\boldsymbol{\tau} | \boldsymbol{\phi}, \mathbf{e}) \Pr(\boldsymbol{\pi} | \boldsymbol{\tau}, \boldsymbol{\phi}, \mathbf{e}) & (12) \\
&= \prod_{i=1}^l \Pr(\phi_i | \phi_1^{i-1}, \mathbf{e}) \times \Pr(\phi_0 | \phi_1^l, \mathbf{e}) \times \\
&\quad \prod_{i=0}^l \prod_{k=1}^{\phi_i} \Pr(\tau_{ik} | \tau_{i1}^{k-1}, \tau_0^{i-1}, \phi_0^l, \mathbf{e}) \times \\
&\quad \prod_{i=1}^l \prod_{k=1}^{\phi_i} \Pr(\pi_{ik} | \pi_{i1}^{k-1}, \pi_1^{i-1}, \tau_0^l, \phi_0^l, \mathbf{e}) \times \\
&\quad \prod_{k=1}^{\phi_0} \Pr(\pi_{0k} | \pi_{01}^{k-1}, \pi_1^l, \tau_0^l, \phi_0^l, \mathbf{e}). & (13)
\end{aligned}$$

Like its predecessor, Equation 13 can be interpreted as describing a generative process. In the first line, we decide the fertility of each word in  $\mathbf{e}$ , followed by the null fertility. In the second line, we choose a tablet for the null word  $e_0$  and each word in  $\mathbf{e}$ . In the third line, we choose a permutation  $\pi_i$  for each tablet  $\tau_i$ ; this will determine the order of these words in  $\mathbf{f}$ . Finally, we choose a permutation for the null word's tablet  $\tau_0$ .

Taken together,  $\boldsymbol{\tau}$  and  $\boldsymbol{\pi}$  completely determine a French string  $\mathbf{f}$  and an alignment  $\mathbf{a}$ . However, the converse is not true; there may be several different pairs  $\boldsymbol{\tau}, \boldsymbol{\pi}$  that correspond to a single pair  $\mathbf{f}, \mathbf{a}$ . We use  $\langle \mathbf{f}, \mathbf{a} \rangle$  to denote the set of all pairs  $\boldsymbol{\tau}, \boldsymbol{\pi}$  that correspond to  $\mathbf{f}, \mathbf{a}$ . Using this notation, we can write

$$\Pr(\mathbf{f}, \mathbf{a} | \mathbf{e}) = \sum_{\boldsymbol{\tau}, \boldsymbol{\pi} \in \langle \mathbf{f}, \mathbf{a} \rangle} \Pr(\boldsymbol{\tau}, \boldsymbol{\pi} | \mathbf{e}) \quad (14)$$

There are  $\phi_i!$  different orders in which we could generate the  $\phi_i$  words of a tablet  $\tau_i$ . Regardless of the order in which we generate these words, there is a permutation  $\pi_i$  that will put these words in the same order as in  $\mathbf{f}, \mathbf{a}$ . Consequently, there are a total of  $\prod_{i=0}^l \phi_i!$  equivalent pairs  $\boldsymbol{\tau}, \boldsymbol{\pi}$  in  $\langle \mathbf{f}, \mathbf{a} \rangle$ .

### 4.3.1 Model 3: Fertility

In addition to the alignment and translation probabilities used in Model 2, Model 3 uses a set of *fertility probabilities*. The notation used is  $n(\phi | e)$ : the fertility of a word depends only on the word itself. We will use the same translation probabilities  $t(f | e)$  as in Model 2, but we replace alignment probabilities  $a(j|i, l, m)$  with *distortion probabilities*  $d(j|i, l, m)$ . In the context of Equation 13, we make three definitions:

$$n(\phi | e_i) \equiv \Pr(\phi | \phi_1^{i-1}, \mathbf{e}) \quad (15)$$

$$t(\tau_{ik} | e_i) \equiv \Pr(\tau_{ik} | \tau_{i1}^{k-1}, \tau_0^{i-1}, \phi_0^l, \mathbf{e}) \quad (16)$$

$$d(\pi_{ik} | i, m, l) \equiv \Pr(\pi_{ik} | \pi_{i1}^{k-1}, \pi_1^{i-1}, \tau_0^l, \phi_0^l, \mathbf{e}). \quad (17)$$

We handle separately the fertility and distortions for the null word  $e_0$ . The purpose of the null word is to account for French words that do not have direct counterparts in the English string; we say these French words are *null-generated*. We now propose a generative mechanism for null fertility. Recall that in Equation 13 our first step was to determine the fertility of each word in  $\mathbf{e}$ . After doing this, we can estimate of the number of words in  $\mathbf{f}$  as  $m' = \sum_{i=1}^l \phi_i$ . Now, with some probability  $p_1$ , we consider adding a null-generated word after each position in  $m'$ . This process is analogous to flipping a weighted coin  $m'$  times, and we can represent it as a binomial distribution. The probability that we add exactly  $\phi_0$  words in this fashion is therefore:

$$\Pr(\phi_0 | \phi_1^l, \mathbf{e}) \equiv \binom{m'}{\phi_0} (1 - p_1)^{m' - \phi_0} p_1^{\phi_0} \quad (18)$$

We call  $p_1$  the *null fertility probability*, and it is a parameter of Model 3.

According to Equation 13, null-generated words (which comprise the tablet  $\tau_0$ ) are permuted in the last step, after all other words have already been assigned positions in  $\mathbf{f}$ . At this point there are exactly  $\phi_0$  positions

with unassigned words, so we can make the following definition:

$$\Pr(\pi_{0k} | \pi_{01}^{k-1}, \pi_1^l, \tau_0^l, \phi_0^l, \mathbf{e}) \equiv \begin{cases} \frac{1}{\phi_0 - (k-1)} & \text{if } \pi_{0k} \text{ is vacant} \\ 0 & \text{otherwise} \end{cases} \quad (19)$$

By this definition, it is clear that the probability of placing the null-generated words in any order is

$$\prod_{k=1}^{\phi_0} \frac{1}{\phi_0 - (k-1)} = \prod_{k=1}^{\phi_0} \frac{1}{k} = \frac{1}{\phi_0!}. \quad (20)$$

Armed with the definitions made in Equations 15–20, we can now rewrite Equation 13:

$$\begin{aligned} \Pr(\boldsymbol{\tau}, \boldsymbol{\pi} | \mathbf{e}) &= \prod_{i=1}^l n(\phi_i | e_i) \times \binom{m - \phi_0}{\phi_0} (1 - p_1)^{m - 2\phi_0} p_1^{\phi_0} \times \\ &\quad \prod_{i=0}^l \prod_{k=1}^{\phi_i} t(\tau_{ik} | e_i) \times \prod_{i=1}^l \prod_{k=1}^{\phi_i} d(\pi_{ik} | i, m, l) \times \frac{1}{\phi_0!}. \end{aligned} \quad (21)$$

If we wish to calculate  $\Pr(\mathbf{f}, \mathbf{a} | \mathbf{e})$ , however, we must do the extra work of summing Equation 21 over all pairs  $\boldsymbol{\tau}, \boldsymbol{\pi} \in \langle \mathbf{f}, \mathbf{e} \rangle$ . Happily, the distortion parameters in Model 3 are such that they assign the same probability to each of these pairs. Recall that there are  $\prod_{i=0}^l \phi_i!$  equivalent pairs  $\boldsymbol{\tau}, \boldsymbol{\pi}$  in  $\langle \mathbf{f}, \mathbf{a} \rangle$ . We can therefore write

$$\begin{aligned} \Pr(\mathbf{f}, \mathbf{a} | \mathbf{e}) &= \prod_{i=0}^l \phi_i! \times \Pr(\boldsymbol{\tau}, \boldsymbol{\pi} | \mathbf{e}) \\ &= \prod_{i=1}^l \phi_i! n(\phi_i | e_i) \times \binom{m - \phi_0}{\phi_0} (1 - p_1)^{m - 2\phi_0} p_1^{\phi_0} \times \\ &\quad \prod_{i=0}^l \prod_{k=1}^{\phi_i} t(\tau_{ik} | e_i) \times \prod_{i=1}^l \prod_{k=1}^{\phi_i} d(\pi_{ik} | i, m, l). \end{aligned} \quad (22)$$

Note that the introduced  $\phi_0!$  cancels the  $\frac{1}{\phi_0!}$  from Equation 21.

It is worth mentioning here that Model 3 is *deficient*, a technical problem

discussed in Section 4.3.3.

### 4.3.2 Model 4: Class-Based Alignment

One of the shortcomings of Model 3 stems from the independence assumptions of its distortion probabilities. Specifically, Model 3 aligns each of the words in  $\mathbf{f}$  independently, without looking at any alignments it has already made. Not only that, but it aligns *positions* in  $\mathbf{e}$  and  $\mathbf{f}$ , without looking at the words in either string. We say that Model 3’s distortion probabilities are *absolute*, because they evaluate connections between specific positions in  $\mathbf{e}$  and  $\mathbf{f}$ . Model 4 addresses these faults by employing a *relative* distortion scheme, where the words in  $\mathbf{f}$  are placed relative to other words rather than assigned absolute positions. In addition, the distortion probabilities in Model 4 depend on the identities of the words being aligned.

Before going further, we must introduce a few more terms. We define the *center*  $c_i$  of a tablet  $\tau_i$  to be the ceiling of the average value of the positions in  $\mathbf{f}$  of its words. Formally, we can write:

$$c_i = \left\lceil \frac{\sum_{k=1}^{\phi_i} \pi_{ik}}{\phi_i} \right\rceil \quad (23)$$

For completeness, we define  $c_0 = 0$ . Next, we define the *head* of  $\tau_i$  to be the word that comes first in  $\mathbf{f}$ , that is, the  $\tau_{ik}$  with the smallest  $\pi_{ik}$ . Finally, we introduce the notion of a *word class*. The idea here is to reduce the number of parameters in our model by subdividing the vocabulary into a small number of classes. There are many statistical methods for learning such classes; the original algorithm is described in [3]. We denote word classes as  $\mathcal{A}(e)$  for English and  $\mathcal{B}(f)$  for French.

At this point, we should make note of a restriction Model 4 places on permutations: the words of a tablet  $\tau_i$  must appear in  $\mathbf{f}$  in left-to-right order. Note that there are no restrictions between the words of  $\tau_i$  and those of any other tablet; this only means that  $\tau_{i1}$  must come before  $\tau_{i2}$ , which must come before  $\tau_{i3}$ , etc. Formally, we require that  $\pi_{ik} < \pi_{ik+1}$  for  $1 \leq i \leq l$  and  $1 \leq k < \phi_i$ . This restriction eliminates all but 1 of the  $\phi_i!$  possible orderings

of  $\tau_i$ . Note that as with Model 3, we handle permutations for the null word separately, so this restriction does not apply to  $\tau_0$ .

In Model 4, we replace the single set of distortion probabilities in Model 3 with two sets. The first set handles placing the head of a tablet:

$$d_1(\pi_{i1} - c_{i-1} | \mathcal{A}(e_i), \mathcal{B}(\tau_{i1})) \equiv \Pr(\pi_{i1} | \pi_1^{i-1}, \tau_0^l, \phi_0^l, \mathbf{e}). \quad (24)$$

Here, distortions are measured relative to the center of the previous tablet. We condition on both the class of the French word being aligned,  $\mathcal{B}(\tau_{i1})$ , as well as the class of the English word it is being aligned with,  $\mathcal{A}(e_i)$ . The second set of distortion probabilities is for placing the rest of the tablet's words:

$$d_{>1}(\pi_{ik} - \pi_{ik-1} | \mathcal{B}(\tau_{ik})) \equiv \Pr(\pi_{ik} | \pi_{i1}^{k-1}, \pi_1^{i-1}, \tau_0^l, \phi_0^l, \mathbf{e}). \quad (25)$$

Here, we measure distortions relative to the position of the previous word of the table, and condition only on the class of the French word being aligned.

Substituting these new distortion probabilities into Equation 21, we arrive at the formula for Model 4:

$$\begin{aligned} \Pr(\boldsymbol{\tau}, \boldsymbol{\pi} | \mathbf{e}) &= \prod_{i=1}^l n(\phi_i | e_i) \times \binom{m - \phi_0}{\phi_0} (1 - p_1)^{m - 2\phi_0} p_1^{\phi_0} \times \\ &\quad \prod_{i=0}^l \prod_{k=1}^{\phi_i} t(\tau_{ik} | e_i) \times \prod_{i=1}^l d_1(\pi_{i1} - c_{i-1} | \mathcal{A}(e_i), \mathcal{B}(\tau_{i1})) \times \\ &\quad \prod_{i=1}^l \prod_{k=2}^{\phi_i} d_{>1}(\pi_{ik} - \pi_{ik-1} | \mathcal{B}(\tau_{ik})) \times \frac{1}{\phi_0!}. \end{aligned} \quad (26)$$

To calculate  $\Pr(\mathbf{f}, \mathbf{a} | \mathbf{e})$  under Model 4, we must recall the restriction we imposed earlier on the order of words on each tablet. Because a unique ordering is defined for every tablet except  $\tau_0$ , under Model 4 there are only  $\phi_0!$  equivalent pairs  $\boldsymbol{\tau}, \boldsymbol{\pi}$  in  $\langle \mathbf{f}, \mathbf{a} \rangle$ . We can thus derive the final form of

Model 4:

$$\begin{aligned}
\Pr(\mathbf{f}, \mathbf{a} | \mathbf{e}) &= \phi_0! \times \Pr(\boldsymbol{\tau}, \boldsymbol{\pi} | \mathbf{e}) & (27) \\
&= \prod_{i=1}^l n(\phi_i | e_i) \times \binom{m - \phi_0}{\phi_0} (1 - p_1)^{m - 2\phi_0} p_1^{\phi_0} \times \\
&\quad \prod_{i=0}^l \prod_{k=1}^{\phi_i} t(\tau_{ik} | e_i) \times \prod_{i=1}^l d_1(\pi_{i1} - c_{i-1} | \mathcal{A}(e_i), \mathcal{B}(\tau_{i1})) \times \\
&\quad \prod_{i=1}^l \prod_{k=2}^{\phi_i} d_{>1}(\pi_{ik} - \pi_{ik-1} | \mathcal{B}(\tau_{ik})). & (28)
\end{aligned}$$

### 4.3.3 Model 5: Non-Deficient Alignment

There is a problem with the distortion probabilities in Models 3 and 4. Let us imagine a degenerate French string  $\mathbf{f}^d$  with two words in the first position, and none in the second position. Recall that in Model 3, we assigned distortion probabilities without regard for positions assigned to earlier words. Because of this, under Model 3 it is perfectly possible that  $\Pr(\mathbf{f}^d | \mathbf{e}) > 0$  for some valid English string  $\mathbf{e}$ . Under Model 4, the situation is even worse: we can place words not only on top of one another, but beyond the boundaries of the string itself. In other words, we can have a string with two words in position  $-2$ ! We call these models *deficient*, because they waste some of their probability on such degenerate strings.

As an aside, we note that the null distortion parameters, modeled separately as they are in Models 3 and 4, do not suffer from this deficiency. This sets the stage for an empirical illustration of the effects of deficiency: when training Models 3 and 4, it is often the case that the null fertility parameter  $p_1$  rises with every iteration. In fact, unless  $p_1$  is capped at a reasonable value, the null word quickly takes over, often to the point where null is aligned to more than half the French words in a typical sentence. Because null distortions are not deficient, a model that uses more of them can assign higher probability to the training data.

The raison d'être of Model 5 is to correct the deficiencies of Models 3 and 4. The general way we accomplish this is by keeping track of all remain-

ing vacant positions in the string at every step. We must also condition the new distortion probabilities on the number of vacancies remaining. Unfortunately, this increases the number of parameters we must learn during training. Whereas in Model 4 we conditioned the head distortion probability  $d_1(\pi_{i1} - c_{i-1} | \mathcal{A}(e_i), \mathcal{B}(\tau_{i1}))$  on both  $\mathcal{A}(e_i)$  and  $\mathcal{B}(\tau_{i1})$ , in Model 5 we are forced to drop the dependence on  $\mathcal{A}(e_i)$ . Because Model 4 includes this dependency, it is actually superior to Model 5 for decoding purposes. Consequently, we will not provide any of the formulas for Model 5 in this paper; for a complete treatment the reader is referred to [5].

#### 4.4 Training

To find  $\Pr(\mathbf{f} | \mathbf{e})$ , we must add up the probabilities of all possible alignments between  $\mathbf{e}$  and  $\mathbf{f}$ :  $\Pr(\mathbf{f} | \mathbf{e}) = \sum_{\mathbf{a}} \Pr(\mathbf{f}, \mathbf{a} | \mathbf{e})$ . In practice, however, the extremely large number of possible alignments makes it impractical to calculate this sum explicitly. A clever reformulation, whereby we switch the order of product and summation, makes it possible to calculate  $\Pr(\mathbf{f} | \mathbf{e})$  exactly for the first two models. For the higher models, however, no such trick is known [5]. Instead, we find the most probable alignment  $\hat{\mathbf{a}}$  and sum  $\Pr(\mathbf{f}, \mathbf{a} | \mathbf{e})$  over a small neighborhood of  $\hat{\mathbf{a}}$ . During decoding, we will simply use  $\Pr(\mathbf{f}, \hat{\mathbf{a}} | \mathbf{e})$  directly.

As the focus of this paper is decoding, we will not discuss training the models; for a complete discussion, the reader is referred to [5].

## Part II

# Decoding

## 5 The Decoding Problem

The models described in the preceding sections tell us how to calculate  $\Pr(\mathbf{f} | \mathbf{e})$  for any pair of strings  $\mathbf{e}$  and  $\mathbf{f}$ . While these models might be very useful to us in solving the translation problem, it is clear that they are not in themselves the whole solution. This is where the decoder enters into the picture. The task of the decoder is, given a previously unseen French string  $\mathbf{f}$  and Translation and Language Models, to find the  $\mathbf{e}$  which maximizes the product  $\Pr(\mathbf{f} | \mathbf{e}) \Pr(\mathbf{e})$ .

The problem of decoding using IBM's translation models is NP-complete. In fact, we can prove the NP-completeness of Model 1 decoding by reducing it to two different problems: a Minimum Set Cover problem and a Hamiltonian circuit (traveling salesman) problem. These two problems highlight two different kinds of complexity inherent in the decoding problem. The first kind of complexity stems from deciding which words to put in a translation, and the second kind from choosing an order for the words. A full treatment of this proof is given in [12].

In the face of this complexity, we have two choices. First, we can simplify our models to make the search problem more tractable. Or second, we can abandon the optimal approach and confine our search to a smaller space. These two approaches have both been tried in various attempts at decoding. The stack-based approach falls solidly into the second category.

### 5.1 Existing Decoders

#### 5.1.1 Stack Decoders

Several other stack decoders for IBM's models have been described in the literature. The first was developed as part of the original Candide project at IBM, and it is after this decoder that the present work is closely modeled.

Although a technical paper on the decoder was said to be forthcoming, the only description ever made public was part of a patent [1]. Unfortunately, the patent only describes a Model 3 decoder, and provides very little in the way of either background or results. Another stack decoder was developed by Wang and Waibel [18]. They describe an interesting heuristic which helps guide the search. However, their decoder only employs Models 1 and 2, and it is unclear how to adapt their heuristic for use with the higher models. There are certain other instances of stack decoders used in machine translation, but to our knowledge none are compatible with IBM Model 4.

### 5.1.2 Other Decoders

Recently, two other types of decoders have been developed, neither based on the stack paradigm. Each takes a very different approach to the problem. Both were developed in parallel with the present work at the Information Sciences Institute at the University of Southern California; for a more detailed discussion the reader is referred to [9].

The first is called the Greedy decoder; it uses a heuristic-repair algorithm to find quick translations. It starts with a simple word-for-word English translation of the French sentence. It then performs simple operations on this prototype translation, such as changing word order, substituting words, adding words, etc. If any of these operations increases the probability of the translation, it modifies the English sentence accordingly, and then tries the operations all over again on the modified sentence. When when none of these operations increase the prototype's probability, the process is halted and the translation returned.

The primary advantage of the greedy decoder is its speed; it operates orders of magnitude faster than other decoders, and its complexity scales polynomially rather than exponentially with sentence length. Its disadvantage is that it examines only a very small portion of the search space. The greedy decoder is quite good at finding literal translations, but if the best translation is far from word-for-word, it can have trouble. Consequently, it performs very well when translating from French to English, but might have

trouble with other language pairs.

The other type of decoder is the Optimal decoder. As the name implies, it is guaranteed to find the optimal translation according to IBM's models. It works by reducing the decoding problem to an integer-programming problem, and then employing commercial software to find a solution. The obvious advantage here is the guarantee of optimality. The disadvantage is that to make the problem tractable it is limited to a bigram language model. Also, it is very slow.

## 6 Stack-Based Decoding

The stack decoding algorithm is a kind of search introduced in the domain of speech recognition [10]. In its original incarnation, a stack decoder essentially performs a limited horizon uniform-cost search using Dijkstra's Algorithm [8]. By building solutions incrementally and storing partial solutions, or hypotheses, in a "stack,"<sup>4</sup> the decoder conducts an ordered search of the solution space. In the ideal case (unlimited stack size and exhaustive search time), a stack decoder is guaranteed to find an optimal solution; our hope is to do almost as well under real-world constraints of limited space and time. The generic stack decoding algorithm follows:

1. Initialize the stack with an empty hypothesis.
2. Pop a hypothesis off the stack and call it  $h$ .
3. If  $h$  is a complete sentence, output  $h$  and terminate.
4. For each possible next word  $w$ , extend  $h$  by adding  $w$  and push the resulting hypothesis onto the stack.
5. Return to 2.

---

<sup>4</sup>The moniker "stack" is somewhat misleading here, because the algorithm is built around what is normally referred to as a priority queue rather than a stack. This priority queue keeps partial solutions (hypotheses) sorted by score.

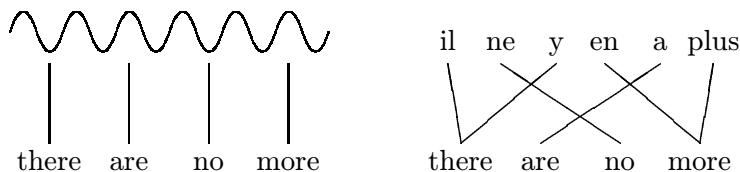


Figure 2: Relationships between decoder input and output for speech recognition (left) and machine translation (right).

When we pop hypotheses off the stack in the same order in which we pushed them on, we are essentially performing a breadth-first search, meaning that we explore all possible paths simultaneously. If we could somehow arrange the stack so that the most promising hypotheses are explored first, we could potentially save ourselves huge amounts of computation. To do this, we need a function which can rank partial hypotheses by their expected score when complete. Put another way, this function should be able to estimate the difficulty of completing a hypothesis, and use this along with its partial score to estimate the hypothesis' value. Such a function is called a *heuristic*; search ordered by such a heuristic is called *A\* search*.

One crucial difference between the decoding process in speech recognition and machine translation is that speech is always produced in the same order as its transcription. Consequently, in speech recognition decoding there is always a simple left-to-right correspondence between input and output sequences. By contrast, in machine translation the left-to-right relation rarely holds even for language pairs as similar as French and English. We address this problem by building the solution from left to right, but allowing the decoder to consume its input in any order. This change makes decoding significantly more complex in machine translation; instead of knowing the order of the input in advance, we must consider all  $n!$  permutations of an  $n$ -word input sentence. This is illustrated in Figure 2.

Moreover, the left-to-right restriction in speech recognition makes it possible to use a simple yet reliable class of heuristics which estimate cost based on the amount of input left to decode. Partly because of the absence of left-to-right correspondence, machine translation heuristics are significantly more difficult to develop [18]. Without a heuristic, a classic stack decoder

is ineffective because shorter hypotheses will almost always look more attractive than longer ones, since as we add words to a hypothesis, we end up multiplying more and more terms to find the probability. Because of this, longer hypotheses will be pushed off the end of the stack by shorter ones even if they are in reality better decodings. Fortunately, by using more than one stack, we can eliminate this effect.

In a multistack decoder, we employ more than one stack to force hypotheses to compete fairly. More specifically, we have one stack for each subset of input words. This way, a hypothesis can only be pruned if there are other, better, hypotheses that represent the same portion of the input. With more than one stack, however, how does a multistack decoder choose which hypothesis to extend during each iteration? We address this issue by simply taking one hypothesis from each stack, but a better solution would be to somehow compare hypotheses from different stacks and extend only the best ones. Another issue faced by multistack decoders is that the number of stacks grows exponentially with sentence length; a solution to this problem might be to combine similar stacks in an intelligent manner. This is discussed in Section 10.

## 7 Operations

At the very heart of a stack decoder are the operations it uses to explore the search space. It is important that we design them with efficiency in mind, as the decoder will spend the vast majority of its time calling these operations millions of times apiece. The formulas presented in Sections 3 and 4 tell us how to calculate  $\Pr(e | \mathbf{f})$  and  $\Pr(e)$ . In a naïve implementation, we might calculate these probabilities from scratch each time we extend a hypothesis. Upon closer inspection, however, we would find our simple algorithm spending most of its time performing the same calculations over and over again. The operations presented here take advantage of the intrinsically incremental nature of A\* search to perform these calculations incrementally.

$\mathbf{e}$	Vector of English words (4.1)
$\mathbf{f}$	Vector of French words (4.1)
$\phi$	Vector of fertilities (4.3)
$\tau$	Vector of tablets (4.3)
$\pi$	Vector of permutations (4.3)
$\mathbf{c}$	Vector of centers of tablets (4.3.2)
$r$	Index of rightmost element of $\mathbf{e}$ (7.1)
$\omega$	Boolean; true if hypothesis is open (7.2)
$p_T$	Translation Model: $\Pr(\mathbf{f}   \mathbf{e})$ (4.3.2)
$p_L$	Language Model: $\Pr(\mathbf{e})$ (3)

Table 1: Fields of a hypothesis, and the section in which each concept was introduced.

## 7.1 Notation

Before delving into the inner workings of the decoder, we must first introduce some terminology. These operations are functions of a hypothesis, denoted  $\mathbf{h}$ , which is defined to be a tuple  $(\mathbf{e}, \mathbf{f}, \phi, \tau, \pi, \mathbf{c}, r, \omega, p_T, p_L)$ . The meaning of each of these fields is given in Table 1. We have already been introduced to all but two of these symbols. The first new symbol,  $r$ , we use to refer to the index of the rightmost word in  $\mathbf{e}$ , so that as  $e_r$  is the rightmost word in  $\mathbf{e}$ . The second,  $\omega$ , is a boolean value which is true if the hypothesis is *open*, and false otherwise. The distinction between open and closed hypotheses is explained in Section 7.2.

## 7.2 Open and Closed Hypotheses

Recall that we build solutions incrementally, from left to right; we only operate on the right end of  $\mathbf{e}$ . Because of this, there are two basic groups of operations with which we can build on a hypothesis: we can *extend* a hypothesis by aligning  $e_r$  to another word in  $\mathbf{f}$ , or we can *add* another English word to  $\mathbf{e}$ , thereby increasing  $r$  by 1. With this in mind, we will distinguish between *open* hypotheses, which we must extend by aligning more words to  $e_r$ , and *closed* hypotheses, where we must add another word

to  $e$ .

The motivation for making this distinction is that the same hypothesis might have very different translation model scores depending on whether it is open or closed. This difference arises from fertility probabilities: in a closed hypothesis, all fertilities are known. In an open hypothesis, however, we know that the fertility of the rightmost English word  $e_r$  is *greater than* the current value of  $\phi_r$ . To this end, we can define a new probability factor:

$$n_{>}(\phi | e) = \sum_{i > \phi} n(i | e). \quad (29)$$

In other words, the  $n_{>}$  factor represents the probability that the fertility of  $e$  is greater than  $\phi$ . For example, say an English word  $e$  almost always has fertility 3, and almost never fertility 1. If we align a single French word to  $e$ ,  $n(1 | e)$  will be very small, so the resulting closed hypothesis will have low probability. However,  $n_{>}(1 | e)$  will be relatively large, so the open hypothesis will have much higher probability. If we only had the closed hypothesis, even if we were allowed to align more words to  $e$ , it would likely stay very far down on the stack, while the open hypothesis has a much better chance of being extended.

### 7.3 Operation Definitions

We can now introduce the six incremental operations available to the decoder: ADD-OPEN, ADD-CLOSED, ADD-ZFERT, ADD-NULL, EXTEND-OPEN, EXTEND-CLOSED. We describe these operations in detail in Sections 7.3.1–7.3.6. These operations are exactly the ones used by the IBM team in their original decoder [1]. It is important to realize that they represent only one of a great many possible means of exploring the search space. There is some justification for choosing these operations, as explained in Section 7.2, but other choices are possible, and could conceivably result in better performance. We will not explore such possibilities here, however.

$r$	$r+1$
$e_r$	$e$
$\phi_r$	$1$
$\tau_{r1}$	$f_j$
$\pi_{r1}$	$j$
$c_r$	$j$
$p_T$	$p_T \times n(1 e) \times t(f_j e) \times d_1(j - c_{r-1}   \mathcal{A}(e), \mathcal{B}(f_j))$
$p_L$	$p_L \times \Pr(e   e_{r-1}, e_{r-2})$

Table 2: Assignments made by  $\text{ADD-CLOSED}(e, j)$

### 7.3.1 Add-closed

The  $\text{ADD-CLOSED}(e, j)$  operation adds  $e$  to  $\mathbf{e}$  and aligns word  $j$  of  $\mathbf{f}$  to it. The resulting hypothesis is closed, so  $e$  cannot subsequently be aligned with any more words in  $\mathbf{f}$ . The changes wrought on  $\mathbf{h}$  are detailed in Table 2. We are adding a word to  $\mathbf{e}$ , so we first increment  $r$ . We then initialize  $e_r$ ,  $\phi_r$ ,  $\tau_{r1}$ ,  $\pi_{r1}$ , and  $c_r$  to reflect the new word’s properties. The calculation of  $p_T$  is precisely what we would expect from Equation 28: we simply multiply together fertility, translation, and distortion probabilities for the new word. Note that only fields that are changed are included in the table. For instance,  $\omega$  must be false when  $\text{ADD-CLOSED}$  is called, because the  $\text{ADD}$  operations only work on closed hypotheses. Because  $\omega$  must also be false in the resulting hypothesis, it is not included in Table 2.

### 7.3.2 Add-open

The operation  $\text{ADD-OPEN}(e, j)$  is, as we might expect, very similar to  $\text{ADD-CLOSED}$ . The difference here is that the resulting hypothesis is open, so the next operation on it must be an  $\text{EXTEND}$  rather than an  $\text{ADD}$  operation. There are two differences from  $\text{ADD-CLOSED}$ . The first difference is that  $\omega$  must be clearly set to *true* in  $\text{ADD-OPEN}$ . Second, we substitute  $n_{>}(1|e)$  for  $n(1|e)$  as the fertility factor. The assignments made by  $\text{ADD-OPEN}$  are given in Table 3.

$r$	$r+1$
$e_r$	$e$
$\phi_r$	$1$
$\tau_{r1}$	$f_j$
$\pi_{r1}$	$j$
$c_r$	$j$
$\omega$	$true$
$p_T$	$p_T \times n_{>}(1 e) \times t(f_j e) \times d_1(j - c_{r-1}   \mathcal{A}(e), \mathcal{B}(f_j))$
$p_L$	$p_L \times \Pr(e   e_{r-1}, e_{r-2})$

Table 3: Assignments made by ADD-OPEN( $e, j$ )

$r$	$r+1$
$e_r$	$e$
$\phi_r$	$0$
$c_r$	$c_{r-1}$
$p_T$	$p_T \times n(0 e)$
$p_L$	$p_L \times \Pr(e   e_{r-1}, e_{r-2})$

Table 4: Assignments made by ADD-ZFERT( $e$ )

### 7.3.3 Add-zfert

This operation is simpler than the previous two because we don't have to worry about  $\tau$  or  $\pi$ , not to mention translation and distortion probabilities. Note that this is an ADD operation, so it must operate on a closed hypothesis. Enforcing this restriction is to our advantage because it reduces the number of ways we can produce identical hypotheses. Note that if  $e_i$  is a zero-fertility word ( $\phi_i = 0$ ), we define  $c_i = c_{i-1}$ ; this is to maintain consistency for the ADD operations. The details of ADD-ZFERT are given in Table 4.

In practice, however, we never perform ADD-ZFERT by itself; it is always immediately followed by an ADD-OPEN or ADD-CLOSED. Doing so allows us to calculate beforehand whether zero-fertility words will improve the probability of a hypothesis. This optimization is discussed in detail in Section

$$\begin{array}{l}
\phi_0 \\
\tau_{0\phi_0} \\
\pi_{0\phi_0} \\
p_T
\end{array}
\left\| \begin{array}{l}
\phi_0 + 1 \\
f_j \\
j \\
p_T \times \frac{C(m-\phi_0|\phi_0)}{C(m-(\phi_0-1)|\phi_0-1)} \times \frac{p_1}{1-p_1} \times t(f_j|e_0)
\end{array} \right.$$

Table 5: Assignments made by ADD-NULL( $j$ )

8.1.1.

### 7.3.4 Add-null

This operation aligns a French word to the null word  $e_0$ . We do not need to change the language model probability  $p_L$  because we are not changing  $\mathbf{e}$  at all. What we need to do is increment  $\phi_0$  and change the combinatorial term in  $p_T$  to reflect its new value. The assignments made by ADD-NULL are given in Table 5.

Although it does not add another word to  $\mathbf{e}$ , we call this operation ADD-NULL rather than EXTEND-NULL because it is only defined on closed hypotheses. Note that this is not really a restriction at all, because Model 4 ignores the order in which null-generated words are produced. Actually, because null-generated words can be produced at any position between English words, there are  $\frac{m!}{(m-\phi_0)!}$  different ways we could produce any given English string, simply by varying the order in which we produce its null-generated words. In Section 8.2 we discuss ways to deal with this redundancy.

### 7.3.5 Extend-open

The two EXTEND operations are mostly simple, with the significant caveat of updating distortion probabilities. The first, EXTEND-OPEN, starts by incrementing  $\phi_r$ , and sets  $\tau_{r\phi_r}$  and  $\pi_{r\phi_r}$  to reflect the new word aligned with  $e_r$ . We must also recalculate the centroid of  $\boldsymbol{\tau}_r$ . Finally, we must update the translation model score  $p_T$ . The correct expression here looks similar to the one for ADD-OPEN; we multiply the probability that the fertility of  $e_r$

$$\begin{array}{l}
\phi_r \\
\tau_r \phi_r \\
\pi_r \phi_r \\
c_r \\
p_T
\end{array}
\left\| \begin{array}{l}
\phi_r + 1 \\
f_j \\
j \\
\left\lceil \frac{\sum_{k=1}^{\phi_r} \pi_{rk}}{\phi_r} \right\rceil \\
p_T \times \frac{n_{>}(\phi_r | e_r)}{n_{>}(\phi_r - 1 | e_r)} \times t(f_j | e_r) \times \text{DPROB}(j)
\end{array} \right.$$

Table 6: Assignments made by EXTEND-OPEN( $j$ )

```

function DPROB( $j$ )
   $a := \phi_{r1}$ 
   $b := \phi_{r1}$ 

  for  $k = 2 \dots (\phi_r - 1)$ 
    if [ $j < a$  and  $\pi_{rk} < a$ ] or [ $j > a$  and  $j > \pi_{rk}$  and  $\pi_{rk} > a$ ]
      then  $a := \pi_{rk}$ 
    if [ $j > b$  and  $\pi_{rk} > b$ ] or [ $j < b$  and  $j < \pi_{rk}$  and  $\pi_{rk} < b$ ]
      then  $b := \pi_{rk}$ 
  end for

  if  $b < j$ 
  then return  $d_{>1}(j - a | \mathcal{B}(f_j))$ 
  if  $b > j$  and  $a > j$ 
  then return  $\frac{d_1(j - c_{r-1} | \mathcal{A}(e_r), \mathcal{B}(f_j)) \cdot d_{>1}(a - j | \mathcal{B}(f_a))}{d_1(\pi_{ra} - b_{r-1} | \mathcal{A}(e_r), \mathcal{B}(f_a))}$ 
  if  $b > j$  and  $a < j$ 
  then return  $\frac{d_{>1}(j - a | \mathcal{B}(f_j)) \cdot d_{>1}(b - j | \mathcal{B}(f_b))}{d_{>1}(b - a | \mathcal{B}(f_b))}$ 

```

Table 7: Definition of the DPROB function. A brief explanation of the variable names is helpful here. First, note that  $e_r$  is the last word in  $\mathbf{e}$ , and DPROB will always be called with  $j = \pi_{r\phi_r}$ , the index in  $\mathbf{f}$  of the latest word produced by  $e_r$ . Also,  $a$  and  $b$  are the indices in  $\mathbf{f}$  of the words produced by  $e_r$  whose positions are closest to  $j$ . That is,  $a$  is the largest value of  $\pi_{rk}$  where  $\pi_{rk} < j$ , and  $b$  is the smallest value of  $\pi_{rk}$  where  $\pi_{rk} > j$ .

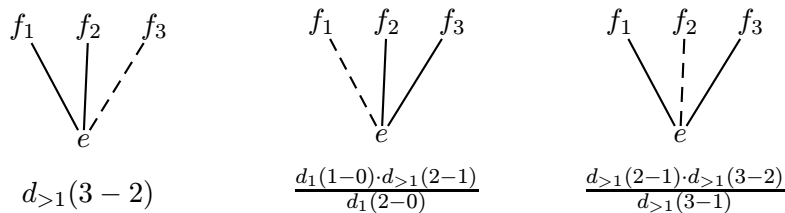


Figure 3: Three cases for distortion probabilities after an EXTEND operation.

is greater than  $\phi_r$  by the translation probability of  $f_j$  given  $e_r$ . The DPROB function calculates the distortion probability; it is discussed below. Note that we need not make any changes to  $e$  or  $r$ , as we are not adding a new English word. For the same reason, there is no need to update the language model score  $p_L$ .

Calculating the distortion probability proves to be by far the most difficult task of the EXTEND operations. The reason for this complexity is that we allow tablets to be built in any order, whereas the distortion calculations in Model 4 require that the words of a tablet appear in  $\mathbf{f}$  in left-to-right order (Section 4.3.2). Because the EXTEND operations defined here can create tablets that do not obey this restriction, we must be careful when calculating the distortion probabilities. The DPROB function handles this task; its operation is described below, and pseudo-code is given in Table 7.

There are three cases we must consider when aligning a new word  $f_j$  to the rightmost English word  $e_r$ . First we have the simple case, where all other words on  $e_r$ 's tablet  $\tau_r$  are to the left of  $f_j$ . Here, we can simply multiply  $p_T$  by the  $d_{>1}$  probability for  $f_j$ . In the second case,  $f_j$  is to the left of all other words on the tablet  $\tau_r$ . In this case,  $f_j$  becomes the new head of the tablet, and we must accordingly multiply by a  $d_1$  probability. Let us call the previous head of  $\tau_r$   $f_h$ . The problem here is that  $p_T$  already contains a  $d_1$  probability factor for  $f_h$ . We must therefore divide out this  $d_1$  probability, and multiply in a new  $d_{>1}$  factor reflecting the new status of  $f_h$  as a non-head word. The third case occurs when  $f_j$  is inserted between two words on the tablet; let us call these two words  $f_a$  and  $f_b$ . Here,  $p_T$  already contains a  $d_{>1}$  factor for  $f_b$ . We must divide this factor out and replace it with two other  $d_{>1}$  factors, one for  $f_j$ , and a new one for  $f_b$  that takes  $f_j$

$$\begin{array}{l}
\phi_r \\
\tau_{r\phi_r} \\
\pi_{r\phi_r} \\
c_r \\
\omega \\
p_T
\end{array}
\left\| \begin{array}{l}
\phi_r + 1 \\
f_j \\
j \\
\left\lceil \frac{\sum_{k=1}^{\phi_r} \pi_{rk}}{\phi_r} \right\rceil \\
false \\
p_T \times \frac{n(\phi_r | e_r)}{n_{>}(\phi_r - 1 | e_r)} \times t(f_j | e_r) \times \text{DPROB}(r, \phi_r)
\end{array} \right.$$

Table 8: Assignments made by EXTEND-CLOSED( $j$ )

into account. These three cases are illustrated in Figure 3.

### 7.3.6 Extend-closed

There are only two differences between EXTEND-CLOSED and EXTEND-OPEN. First, the result of an EXTEND-CLOSED operation is closed, so we must set  $\omega$  to *false*. Second, we substitute an  $n$  factor for the  $n_{>}$  factor when calculating  $p_T$ . In all other ways, the two EXTEND operations are identical.

## 7.4 Sample Decoding

Perhaps the best way to understand the operations we have just described is to see them in action. To this end, we in Figure 4 we present a simple example that illustrates most of the operations.

# 8 Optimizations

## 8.1 Operation Complexity

Of the operations, EXTEND-OPEN, EXTEND-CLOSED, and ADD-NUL are the least expensive, as we must only consider each of the words in  $\mathbf{f}$  which are still unaligned. The operations ADD-OPEN and ADD-CLOSED are more expensive, as we must consider not only each unaligned French word in the hypothesis, but all possible English translations of each one. Assuming a

voulez - vous vous taire ! --> you shut up !

- 0) none                   empty  
    $p = p_0^6 * C(6,0) = 0.885842$
- 1) add-open               you+/4  
    $p * n(>1|you) * t(vous|you) * d(4|0,11) = 1.40866e-03$
- 2) extend-open        you+/4 you+/2  
    $p * n(>2|you) / n(>1|you) * t(-|you) * d(2|0,3) * d(2|-1,11) /$   
       $d(4|0,11) = 3.47e-04$
- 3) extend-open        you+/4 you+/2 you+/1  
    $p * n(>3|you) / n(>2|you) * t(voulez|you) * d(1|0,21) *$   
       $d(1|-1,3) / d(2|0,3) = 2.40e-07$
- 4) extend-closed    you+/4 you+/2 you+/1 you/3  
    $p * n(4|you) / n(>3|you) * t(vous|you) * d(1|-1,11) * d(1|-1,11) /$   
       $d(2|-1,11) = 3.84e-07$
- 5) add-closed         you+/4 you+/2 you+/1 you/3 shut/5  
    $p * n(1|shut) * t(taire|shut) * d(2|33,10) = 1.80467e-07$
- 6) add-zfert         you+/4 you+/2 you+/1 you/3 shut/5 up/-1  
    $p * n(0|up) = 1.4009e-09$
- 7) add-closed         you+/4 you+/2 you+/1 you/3 shut/5 up/-1 !/6  
    $p * n(1|!) * t(!|!) * d(1|44,18) = 8.00002e-10$

Figure 4: A sample decoding; the formatting and probabilities are taken from an actual session. Note that you+/2 indicates that you is still open, and is aligned to word 2 of the French sentence—in this case the hyphen. Note also the word classes used in distortion probabilities; a class of -1 indicates a non-head distortion.

vocabulary size of 40,000 words, a naïve implementation would make ADD-OPEN and ADD-CLOSED 40,000 times as expensive as the EXTEND operations.

In practice, a simple way to reduce the cost of the ADD operations is to restrict the number of English words we consider as translations for each French word. We will therefore only consider the ten most likely *inverse translations* of each French word. We use the term inverse translations because the translation model usually works in the other direction. This restriction makes ADD-OPEN and ADD-CLOSED only ten times more expensive than the EXTEND operations.

### 8.1.1 Zero-Fertility Operations

Because it is always performed in conjunction with another ADD operation, ADD-ZFERT is by far the most expensive operation. With an English vocabulary size of 40,000, a naïve implementation of ADD-ZFERT would be  $40,000 \times 40,000 = 1,600,000,000$  times as expensive as EXTEND! Clearly, we must find some way to reduce this cost. We can certainly use the shortcut suggested above to speed up ADD-OPEN and ADD-CLOSED. Since zero-fertility words are not aligned to any French words, however, we cannot choose them from a list of inverse translations; we must use some other mechanism to suggest them.

To find out what makes a successful zero-fertility word, we might first look at the definition of ADD-ZFERT in Table 4. We see that it affects the score of a hypothesis in two ways: we multiply the translation model score by  $n(0|e)$ , and the language model score by  $\Pr(e|e_{r-1}, e_{r-2})$ . For the operation to be beneficial, both these values should be as large as possible. Although the trigram score  $\Pr(e|e_{r-1}, e_{r-2})$  will clearly depend on the context, we can use a word’s unigram probability  $\Pr(e)$  as an estimate of its trigram score in arbitrary contexts (Section 3). We could therefore imagine that the product  $n(0|e)\Pr(e)$  would provide a reasonable estimate of a word’s potential as fodder for ADD-ZFERT, and rank all English words according to this score. When performing an ADD-ZFERT operation, then, we can limit ourselves to words from the top of this list. In practice, we use  $n(0|e)^2\Pr(e)$  to rank

$n(0 e)^2 \Pr(e)$	$e$	$n(0 e)$	$\Pr(e)$
0.009215	to	0.56	0.0286
0.003123	be	0.62	0.0080
0.002834	hon.	0.85	0.0039
0.002792	a	0.41	0.0159
0.002758	,	0.29	0.0319
0.002561	is	0.36	0.0189
0.002515	of	0.29	0.0298
0.002446	in	0.36	0.0185
0.001858	are	0.50	0.0073
0.001625	for	0.42	0.0088
0.001581	will	0.51	0.0059
0.001579	on	0.50	0.0063
0.001451	would	0.58	0.0042
0.001389	do	0.65	0.0032
0.001293	the	0.14	0.0626
0.001238	with	0.50	0.0048
0.001155	—	0.56	0.0035
0.001032	by	0.50	0.0040
0.001002	as	0.44	0.0050
0.000988	have	0.34	0.0082
0.000962	out	0.82	0.0014
0.000871	been	0.52	0.0031
0.000842	that	0.20	0.0193
0.000829	per	0.82	0.0012
0.000687	there	0.40	0.0041
0.000670	up	0.77	0.0011
0.000588	it	0.23	0.0108
0.000572	at	0.41	0.0033
0.000484	an	0.41	0.0028
0.000435	hear	0.48	0.0018
0.000419	being	0.66	0.0009
0.000413	made	0.55	0.0013
0.000372	about	0.39	0.0023
0.000367	does	0.54	0.0012

Table 9: Top zero-fertility words in the Hansards. Note that “hon.” ranks third on this list, a clear reflection of the corpus’s parliamentary origins.

zero-fertility words, which gives a heavier weight to the fertility factor. Table 9 presents the top zero-fertility words from the Hansards.

While restricting the list of candidates in this way improves matters, ADD-ZFERT remains orders of magnitude more expensive than the other operations. There seem to be two choices: either accept the cost of ADD-ZFERT as inevitable, or restrict the list of candidate words to the point where the operation is all but useless. Happily, we can do better than either of these options. The answer is that we can tell in advance whether a zero-fertility word will increase the probability of a hypothesis. According to the definition of the decoding problem, a zero-fertility English word can only make a decoding more likely by increasing  $\Pr(\mathbf{e})$  more than it decreases  $\Pr(\mathbf{f}, \mathbf{a} | \mathbf{e})$ . We know that adding a zero-fertility word will decrease  $\Pr(\mathbf{f}, \mathbf{a} | \mathbf{e})$  because it adds a term  $n(0 | e_i) < 1$  to the calculation. If we are using bigrams as the language model, this simply means that a zero-fertility word  $e_i$  will be beneficial between two other English words  $e_{i-1}$  and  $e_{i+1}$  if and only if

$$n(0 | e_i) < \Pr(e_i | e_{i-1}) \Pr(e_{i+1} | e_i). \quad (30)$$

In this way, we can generate a list of zero-fertility words specifically for  $e_{i-1}$  and  $e_{i+1}$ , and include only words which will improve the overall score of the hypothesis. Moreover, Equation 30 depends only on the three English words, so it can be calculated once and then cached for later usage.

By only considering helpful zero-fertility insertions, we save ourselves enormous amounts of wasted calculation, in many cases eliminating all candidates and reducing the cost of ADD-ZFERT to less than that of ADD-NULL.

## 8.2 Redundant Hypotheses

One drawback of the operations delineated in Section 7 is that it is possible for two different sequences of operations to produce identical results. This is undesirable because the identical hypotheses take space on the stacks, disproportionately displacing other valid hypotheses. There are two ways this redundancy can arise: ADD-NULL and the EXTEND operations. The problem with these operations is that they are not order-sensitive; they can

be applied in any order without affecting the final translation. For instance, if a single French word is null-generated, it makes no difference whether the corresponding ADD-NULL operation is performed first, last, or anywhere in between. Similarly, if two French words are to be aligned with a single English word, the order in which we perform the EXTEND operations to align them makes no difference whatsoever.

The best way to deal with this problem would be to eliminate the potential for redundancy. For instance, we might force EXTEND operations to consume French words in left-to-right order. We could also force ADD-NULL operations to consume French words left-to-right, and to ensure they are only applied at the end of the sequence, after all other operations. In the present work, however, we do not deal with this problem in a rigorous way. Rather than treating the disease, we treat the symptom. Specifically, we assume that non-redundant hypotheses will have different probability values, and delete all but one hypothesis with each probability. The odds are very low that two distinct hypotheses be assigned exactly the same language model probability and translation model probability, and informal testing seemed to bear this out. To be sure, it is likely that there will be some duplication, and at some point a valid hypothesis will no doubt be inappropriately discarded. However, even if this is the case, recall that the problem arises only when the hypotheses have exactly the same probability to begin with; the discarded one will likely be only marginally better, if at all.

### 8.3 Hypothesis Data Structure

The simplest data structure we could use to represent a hypothesis would store all information relevant to the hypothesis. That is, each hypothesis would contain all the fields listed in Table 1. Given the incremental nature of our operations, however, we might end up duplicating a large amount of information each time we apply an operation to a hypothesis. When we perform an operation on a hypothesis to produce a new hypothesis, let us call the original hypothesis the parent and the new one the child. A hypothesis will clearly share some of its features, but not all of them,

Old	New	Description
$e$	$e$	Rightmost English word
$f$	$e_{-1}, e_{-2}$	Previous two English words
$\phi$	$\phi_r$	Current fertility of $e$
$\tau$	$\phi_0$	Current fertility of $e_0$
$\pi$	$\pi$	Most recent alignment of $e$
$c$	$c$	Centroid of $e$
$r$	$par$	Pointer to parent hypothesis
$\omega$	$\omega$	Boolean; true if hypothesis is open
$p_T$	$p_T$	Translation Model probability
$p_L$	$p_L$	Language Model probability

Table 10: An efficient hypothesis representation.

with its children. If we could allow its children access to these features, there would be no need for duplicate copies of this information. If, with each hypothesis, we store a link to its parent, we can accomplish this. By our definition, each hypothesis can only have one parent, as it can only be created once. However, it can have many children, because we can try many different operations on it. This means that our hypothesis data structure will take the form of a massively branched tree, with complete hypotheses at the leaves and the empty hypothesis at the root. The content of a complete hypothesis will be read by starting at a leaf and following the parent links all the way to the root, collecting information at every node along the way.

Table 10 defines exactly such an incremental data structure. As is illustrated by the side-by-side comparison in Table 10, the new structure has similarities and differences with the original definition in Table 1. We examine each variable in turn. First, we need only store one English word  $e$  with each hypothesis; the rest of  $e$  can be found traversing the tree through a hypothesis' parents. Because the value of  $f$  never changes during a single decoding, we assume that it can be made globally accessible, and need not be stored with each hypothesis. It is useful to store  $e_{-1}$  and  $e_{-2}$  with each hypothesis, because these values are needed for the language model calculation. It would be possible to obtain them by traversing the tree, but is not

trivial as we must skip both null and open hypotheses. We can replace the vector  $\phi$  with two values,  $\phi_r$  and  $\phi_0$ ; the rest of the fertility values will not be needed for future calculations. The vector of tablets  $\tau$  is implicit in the values of  $\mathbf{f}$  and  $\boldsymbol{\pi}$ , and is not needed in future calculations. We need only store a single  $\pi$  and  $c$  with each hypothesis; the rest of  $\boldsymbol{\pi}$  and  $\mathbf{c}$  is stored in the parents. We do not need the value of  $r$ , because we only use relative distortions that are independent of position in  $\mathbf{e}$ . Instead, we store *par*, a pointer to the parent hypothesis used when traversing the tree. The final three values,  $\omega$ ,  $p_T$ , and  $p_L$ , are the same in the old and new definitions.

By this incremental definition, hypotheses are much smaller than they would otherwise be. Furthermore, their size will not depend on sentence length.<sup>5</sup> In fact, hypotheses under the new definition will be approximately a factor of  $n/2$  smaller than under the old definition, where  $n$  is the number of operations in the hypothesis. Each hypothesis is 68 bytes in the present implementation of the decoder, which means that approximately 16 million can be stored in a gigabyte. For a 10-word sentence, this translates to 15,000 per stack. For a 15-word sentence, however, we can only have 480 per stack. Such is the vicious exponential nature of using multiple stacks.

In truth, it might be possible to further pare down the size of hypotheses. For instance, not all fields are used in all hypotheses: the  $e$  field is unnecessary in open hypotheses, and the  $\phi$  field unnecessary in closed ones, and we could perhaps get by without  $e_{-1}$  and  $e_{-2}$ . Making these changes would provide marginal improvement, perhaps at the cost of reduced performance.

## 9 Results

The results given in this section are taken from [9], where the performance of the stack decoder is compared to that of two other decoders: one greedy and one optimal, as described in Section 5.1.2. The test set contained 505 French sentences, uniformly distributed over the lengths 6, 8, 10, 15, and 20. The three decoders were evaluated with respect to speed, search optimality,

---

<sup>5</sup>To be fair, a longer hypothesis will have more parents, but these parents will be shared with other hypotheses.

and translation accuracy. Note that because Model 4 is not a perfect model of the translation process, even an optimal search might produce a flawed translation.

Each decoding was classified into one of six categories. If we call the optimal decoding  $e$  and the decoder’s output  $\hat{e}$ , the categories are:

- No Error (NE):  $e = \hat{e}$ , and  $\hat{e}$  is a perfect translation.
- Pure Model Error (PME):  $e = \hat{e}$ , but  $\hat{e}$  is not a perfect translation.
- Deadly Search Error (DSE):  $e \neq \hat{e}$ , and while  $e$  is a perfect translation,  $\hat{e}$  is not.
- Fortuitous Search Error (FSE):  $e \neq \hat{e}$ , and  $\hat{e}$  is a perfect translation, while  $e$  is not.
- Harmless Search Error (HSE):  $e \neq \hat{e}$ , but both  $e$  and  $\hat{e}$  are perfect translations.
- Compound Error (CE):  $e \neq \hat{e}$ , and neither is a perfect translation.

Here, we define a “perfect” translation as one judged by a human to transmit the entire meaning of the source sentence using flawless target-language syntax. In addition, we can define two more general categories:

- Search Error (SE):  $e \neq \hat{e}$ . DSE or FSE or HSE or CE. The decoder produces a translation that is not optimal according to Model 4. Note that the Optimal decoder is immune to Search Error by definition.
- Translation Error (TE):  $\hat{e}$  is not a perfect translation. PME or DSE or CE.

Table 9 demonstrates that the stack decoder performs a much more complete search than the greedy decoder, at least for short sentences. It suffers 5 and 20 search errors at lengths of 6 and 8, compared to 18 and 43 for the greedy decoder. Moreover, only 2 of these 25 search errors (8%) are deadly search errors, meaning that in most cases where the stack decoder misses an optimal solution, the optimal solution is not a perfect translation

L <sup>a</sup>	Decoder	Time <sup>b</sup>	SE	TE	NE	PME	DSE	FSE	HSE	CE
6	optimal	47.50	0	57	44	57	0	0	0	0
6	stack	0.79	5	58	43	53	1	0	0	4
6	greedy	0.07	18	60	38	45	5	2	1	10
8	optimal	499.00	0	76	27	74	0	0	0	0
8	stack	5.67	20	75	24	57	1	2	2	15
8	greedy	2.66	43	75	20	38	4	5	1	33

Table 11: Comparison of decoders on sets of 101 test sentences, using a bigram language model.

---

<sup>a</sup>Sentence length.

<sup>b</sup>Seconds per sentence.

to begin with. By contrast, 9 of the 61 greedy decoder search errors (15%) are deadly. As an aside, we should note the high number of fortuitous search errors for the greedy decoder: 7, as opposed to 2 for the stack decoder. This seems to be a result of the greedy decoder’s bias towards word-for-word translations: it is rewarded for its limited search! With a language pair more diverse than French and English, this advantage would likely be less pronounced.

Table 12 compares the greedy and stack decoders at various sentence lengths, using a trigram language model. First of all, note the reduced number of translation errors—in one case from 57 to 42—that we can attribute to using a trigram rather than bigram language model. Note that the stack decoder has a pronounced advantage in accuracy at lengths up to 10. At length 15, however, this advantage all but disappears. At this length, there are two factors limiting the performance of the stack decoder. First of all, the performance of Model 4 drops off quickly with sentence length, so there is less room for a better search to show improvements in sentence accuracy. Second of all, at length 15 there are  $2^{15}$  stacks, so even a gigabyte of memory provides space for less than 500 hypotheses per stack; at this point hard pruning may be a problem.

To sum up, the advantage of the stack decoder is that it explores a much larger portion of the search space than the greedy decoder, while running

L	Decoder	Time (s)	TE
6	stack	13.72	42
6	greedy	1.58	46
6	greedy*	0.07	46
8	stack	45.45	59
8	greedy	2.75	68
8	greedy*	0.15	69
10	stack	105.15	57
10	greedy	3.83	63
10	greedy*	0.20	68
15	stack	> 2000	74
15	greedy	12.06	75
15	greedy*	1.11	75
15	greedy1	0.63	76
20	greedy	49.23	86
20	greedy*	11.34	93
20	greedy1	0.94	93

Table 12: Comparison of decoders using a trigram language model. Note that greedy\* and greedy1 are less powerful versions of the greedy decoder optimized for speed.

faster than the optimal decoder. Also, it can employ trigrams in the language model, while the optimal decoder is limited to bigrams. The disadvantage is that its time and space complexity are exponential in the length of the input sentence.<sup>6</sup>

## 10 Future Work

The work described here only represents the foundation of a stack decoder. There are an enormous number of improvements, both major and minor, that could be made to the decoder.

A very interesting experiment would be to use the stack and greedy decoders in tandem. Specifically, the stack decoder could first be used to produce an n-best list of translations. The greedy decoder could then be used to explore the neighborhoods of each of the translations on this list. Using the decoders in this way exploits their individual strengths: the stack decoder explores a much larger portion of the search space, while the greedy decoder excels at making incremental improvements to translations as a whole. It seems likely that the results of such a composite decoder would be much better than those of the individual components.

Perhaps the most significant single improvement that could be made to the stack decoder would be the incorporation of an accurate heuristic. This would help guide the search in a much more structured manner, and potentially save enormous amounts of computation that is currently wasted on poor hypotheses. Even a heuristic that is only marginally accurate would help if it allowed us to decide which stacks are more promising. Unfortunately, devising such a heuristic is a very difficult problem. The IBM team used a very simple one based on unigram frequencies of the French words. This heuristic depends on the assumption that frequent French words are easier to translate than rare French words; how beneficial it would prove in practice is unknown.

Another potential improvement was described in Section 8.2: detecting

---

<sup>6</sup>Space complexity is exponential if stack size is kept constant; this can be offset by increasing pruning.

redundant hypotheses as they are pushed onto the stack is not a very good solution for the problem. The ways for eliminating the redundancy suggested in the same section might prove to be a much better fix to this problem.

A different kind of improvement would be a better allocation of stack sizes. In the current version, all stacks are the same size. However, because of the nature of subsets, there are always many more subsets of some lengths than others. For example, for a sentence of length 6, there are 20 different subsets of size 3, but only 6 of size 5. This means that the decoder will store more than three times as many hypotheses of length 3 than of length 5. It is unclear how this discrepancy affects the decoder's performance. A related idea would be to reduce the number of stacks overall by combining "similar" stacks. Similarity could be judged by a heuristic; it would be interesting to try using IBM's French unigram heuristic. In this way we might reduce the number of stacks, increasing the amount of space allocated to each one, and hopefully reduce the effect of hard pruning. This would also improve the decoder's running speed, as we would have fewer hypotheses to extend during each iteration.

## 11 Conclusion

In this paper, we have provided an introduction to statistical machine translation, and a detailed description of IBM Models 1–4. We have also described in detail the implementation of a stack decoder for Model 4, and several issues which arise in such an implementation. We have given some results comparing the speed and quality of this stack decoder and several other decoders. Finally, we have made a detailed list of areas for future work on stack decoding. Our hope is that anyone interested in stack decoding for statistical machine translation will be able to use this work as a baseline.

## References

- [1] Brown, Peter F., John Cocke, Stephen A. Della Pietra, Vincent J. Della Pietra, Fredrick Jelinek, Jennifer C. Lai, and Robert L. Mercer. 1995. Method and system for natural language translation. United States Patent 5,768,603.
- [2] Berger, Adam L., Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, John R. Gillett, John D. Lafferty, Robert L. Mercer, Harry Printz, Luboš Ureš. 1994. The Candide system for machine translation. *Human Language Technology*, pp. 157–162.
- [3] Brown, Peter F., John Cocke, Stephen A. Della Pietra, Vincent J. Della Pietra, Fredrick Jelinek, John D. Lafferty, Robert L. Mercer, and Paul S. Roossin. 1990. A statistical approach to machine translation. *Computational Linguistics*, 16(2).
- [4] Brown, Peter F., Stephen A. Della Pietra, Vincent J. Della Pietra, Meredith J. Goldsmith, Jan Hajic, Robert L. Mercer, Surya Mohanty. 1993. But dictionaries are data too. *Human Language Technology*, pp 202–205.
- [5] Brown, Peter F., Stephen A. Della Pietra, Vincent J. Della Pietra, Robert L. Mercer. 1993. The mathematics of statistical machine translation: parameter estimation. *Computational Linguistics*, 19(2).
- [6] Church, Kenneth W. 1993. *Char\_align*: a program for aligning parallel texts at the character level. In *ACL 31*.
- [7] Davis, Mark W., Ted E. Dunning, and William C. Ogden. 1995. Text alignment in the real world: improving alignments of noisy translations using common lexical features, string matching strategies and n-gram comparisons. In *Proceedings of the European Association for Computational Linguistics*.
- [8] Dijkstra, E. W. A note on two problems in connection with graphs. *Numerical Mathematics*, (1).

- [9] Germann, Ulrich, Michael Jahr, Kevin Knight, Daniel Marcu, and Kenji Yamada. 2001. Fast decoding and optimal decoding for machine translation. In *ACL 39*.
- [10] Jelinek, Frederick. 1969. A fast sequential decoding algorithm using a stack. *IBM Research Journal of Research and Development*, 13.
- [11] Knight, Kevin and Jonathan Graehl. 1997. Machine transliteration. In *Proceedings of ACL 35/EACL 8*.
- [12] Knight, Kevin. 1999. Decoding complexity in word-replacement translation models. *Computational Linguistics*, 25(4).
- [13] Manning, Christopher and Hinrich Schütze. 1999. *Foundations of Statistical Natural Language Processing*. Cambridge, Massachusetts: MIT Press.
- [14] Melamed, I. Dan. 1995. Automatic evaluation and uniform filter cascades for inducing n-best translation lexicons. In *Third Workshop on Very Large Corpora*.
- [15] Simard, Michel, George F. Foster, Pierre Isabelle. 1992. Using cognates to align sentences in bilingual corpora. In *Proceedings of the Fourth International Conference on Theoretical and Methodological Issues in Machine Translation*.
- [16] Tiedemann, Jörg. 1999. Automatic construction of weighted string similarity measures. In *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*.
- [17] Tillmann, C., S. Vogel, H. Ney, and A. Zubiaga. 1997. A DP-based search using monotone alignments in statistical translation. In *Proceedings of ACL 32*.
- [18] Wang, Ye-Yi, and Alex Waibel. 1997. Decoding algorithm in statistical machine translation. In *Proceedings of ACL 32/EACL 8*.

- [19] Weaver, Warren. 1955. Translation. In William N. Locke and A. Donald Booth (eds.), *Machine Translation of Languages: Fourteen Essays*, pp. 15–23. New York: John Wiley & Sons.
- [20] Wu, Dekai. 1996. A polynomial-time algorithm for statistical machine translation. In *Proceedings of ACL 31*.