Chinese-English Statistical Machine Translation by Parsing

Yue Zhang
Mansfield College

Computing Laboratory
University of Oxford
2006

Abstract

Statistical machine translation (SMT) has evolved from the word-based level to higher levels of abstraction. Currently the best known systems are phrased-based, and recent research has started to explore tree-based systems with syntactical information. This thesis aims to study large-scale Chinese-English SMT using a syntactic tree-based model. From the engineering point of view, SMT systems are very complex to build. However, existing pieces of software can bring the work load to a manageable level for this thesis. Using the GenPar framework and other software, this thesis studies Chinese-English SMT by parsing by large-scale experiments. This is the first application of GenPar on Chinese-English SMT.

The experiments show that the accuracy of Chinese-English SMT by parsing is comparable to existing SMT by parsing of other language pairs. However, the accuracy of current MT methods is still largely below human translation, and is influenced by the difference between training and testing data, such as the writing style and domain. Two important factors in the SMT by parsing model are studied, and it is observed that though the accuracy of word-to-word alignment influences the translation accuracy, the mono-lingual English and Chinese grammars do not have a significant impact on the results. From the above observations, advantages and weaknesses of the SMT model are analysed, and possible future improvements for Chinese-English SMT are suggested.

This thesis is organised in three main parts. The first chapter presents the introduction and overview of the thesis. The second and third chapters summarise the related theories by literature review, giving a detailed exposition of the theory of SMT and SMT by parsing. The last two chapters report the novel experiments of Chinese-English SMT by generalised parsing. By discussing the experimental output, the last chapter summarises this thesis and proposes further work.

Acknowledgements

I would like to express my gratitude to my wonderful supervisor Dr. Stephen Clark, who has been giving me knowledgeable guidance through the work of this thesis, and thoroughly reviewed my draft many times. I am also thankful to my friends, including Xi Cheng, who has been discussing new ideas with me and giving me inspirations since the beginning of the course, Hugh Gibson, who has been listening to my problems, making kind suggestions, and Xuan Wang, who visited me and gave me much strength.

Most of all, I want to thank my girlfriend Qian, for sharing my happiness and difficulties all the time (and reading the introduction!), and my parents, for supporting my choice and encouraging me as always.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

1.1.1 Machine translation

The idea of machine translation (MT) can be traced back to the seventeenth century, but it became realistically possible only in the middle of the twentieth century (Hutchins, 2005). Soon after the first computers were developed, research began on MT algorithms. The earliest MT systems consisted primarily of large bilingual dictionaries and sets of translation rules. Dictionaries were used for word level translation, while rules controlled higher level aspects such as word order and sentence organisation. Starting from a restricted vocabulary or domain, rule based systems proved useful. But as the study progressed, researchers found that it is extremely hard for rules to cover the complexity of natural language, and the output of the MT systems were disappointing when applied to larger domains. Little breakthrough was made until the late 1980's, when the increase in computing power made statistical machine translation (SMT) based on bilingual language corpora possible. In the beginning, much scepticism about SMT existed from the traditional MT community because people doubted whether statistical methods based on counting and mathematical equations can be used for the sophisticated linguistic problem. However, the potential of SMT was justified by pioneering experiments carried out at IBM in the early 1990s (Brown et al., 1993). Since then the statistical approach has become the dominant method in MT research.

1.1.2 Statistical machine translation

1.1.2.1 Three important factors: training, decoding and the statistical model

SMT is accomplished in two steps. Before translation, statistical tables are built from bilingual corpora containing manual translations. These tables collect statistical information such as the characteristics of well formed sentences, and the correlation between the languages. During translation, the collected statistical information is used to find the best translation for the input sentences. Normally, the statistical table building process is called the *learning* (or *training*) process, while the translation step is called the *decoding* process.

The decoding process for SMT is essentially a search problem (Russell and Norvig, 2003). Given an input sentence, it searches for the best translation by suggesting and giving scores to possible candidates. Take the instance of Chinese-English translation for example. Suppose that a Chinese sentence is expressed as $F_1^J = f_1, ..., f_j, ..., f_J$, while its English translation is expressed as $E_1^I = e_1, ..., e_i, ..., e_I$.[1] In mathematical form, E_1^I and F_1^J satisfies:

$$E_1^I = \arg\max_{E_1^I} SCORE(E_1^I, F_1^J) \qquad (1\text{-}1)$$

In the above equation, $SCORE(E_1^I, F_1^J)$ evaluates how likely the sentence E_1^I is the translation of the sentence F_1^J. It is computed from the statistical information collected in the learning process. Obviously no statistical tables could store $SCORE(E_1^I, F_1^J)$ directly for all possible sentence pairs E_1^I and F_1^J, because they are far too numerous. Hence the score needs to be broken down into computable factors which can be stored. A statistical *model* defines the method to compute $SCORE(E_1^I, F_1^J)$ mathematically, and which factors are stored in statistical tables. It is central to an SMT system.

1.1.2.2 The evolution of SMT models

The pioneer models for SMT are word-based (Brown et al., 1993). These models assume that sentences are translated word by word. Words in an input sentence are translated individually and put in a specific order to make the translation. Moreover, the word alignment between the input and output sentences follows certain patterns. For example, Table 1-1 shows some patterns of word alignments.

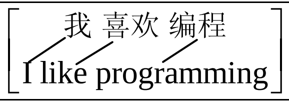
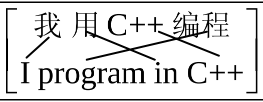
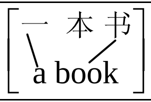
Example			
Alignment	Simple one to one	Reordering	Align to none

Table 1-1: word alignment examples

[1] This thesis follows the conventional use of *e* for English words and *f* for foreign words in the literature.

Based on the above assumption, the word-based models calculate $SCORE(E_1^I, F_1^J)$ for candidate translations by statistical data on the word alignment pattern as well as word-to-word translations. The detailed theory is analysed in Section 2.1.2.

The word-based approaches are comparatively simple and efficient. They proved to be a successful start of SMT approaches. However, the assumption that sentences are translated word by word is oversimplified in some situations, which may lead to potential loss of accuracy in the calculation of $SCORE(E_1^I, F_1^J)$. For example, Table 1-2 shows some Chinese and English sentences that are not strictly translated on the basis of individual words.

Example	我 非常 喜欢 它 I like it very much	令 人 激动 的 旅行 exciting trips	重要 人物 big gun
Alignment	One to many	Many to one	Many to many

Table 1-2: phrase alignment examples

The above examples contain unbreakable phrases as units of translation. To improve the accuracy, such phrases should be included in the model.

Moving from the word-based models, a number of phrase-based models were developed (Koehn et al., 2003). These models separate a sentence into phrases before translation, while they deal with phrase ordering by similar techniques that the word-based models use. Consequently, they calculate $SCORE(E_1^I, F_1^J)$ by statistical data of the phrase alignment and the phrase-to-phrase translations. The theory of phrase-based models is analysed in more detail in Section 2.1.3.

Phrase-based models improved the accuracy over word-based models. However, they did not improve the model of sentence order patterns, which becomes the bottleneck of further improvement. Chiang (2005) used the following example to illustrate the problem of the phrase reordering models:

Original sentence	澳洲 是 与 北 韩 有 邦交 的 少数 国家 之一
Output of a phrase-based SMT system	[Australia] [is] [diplomatic relations] [with] [North Korea] [is] [one of the few countries]
Correct transltion	Australia is one of the few countries that has diplomatic relations with North Korea

Table 1-3: phrase-based translation example; adapted from (Chiang, 2005)

In the above example, the phrase-based SMT system translated “diplomatic relations with North Korea” and “one of the few countries” correctly. However, it failed to put the long phrases in the correct order. One possible reason is that the alignment model is based on flat reordering patterns. It may perform well with local phrase orders, but not as well with long sentences and complex orders.

Based on the above observations, Chiang (2005) developed a hierarchical phrase based model. Chiang’s model evolved from the phrase-based models. However, different from simple phrases, hierarchical phrases have recursive structures. For example, “have diplomatic relation with North Korea” can be viewed as a hierarchical phrase, where “have … with …” embeds “diplomatic relation” and “North Korea”. This structure is illustrated in Figure 1-1.

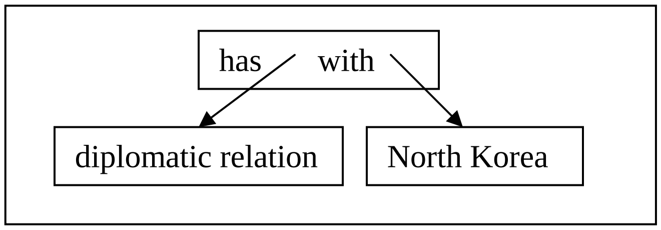


Figure 1-1: the hierarchical phrase structure

Now “与 $\tilde{f}_1$ 有 $\tilde{f}_2$” is translated to “have $\tilde{e}_2$ with $\tilde{e}_1$”, where sub phrases $\tilde{f}_1$ and $\tilde{f}_2$ translate to $\tilde{e}_2$ and $\tilde{e}_1$, respectively. With hierarchical structures, long phrase orders are divided into sub phrase orders, which are much simpler. Figure 1-2 shows a longer example of hierarchical phrase translation.

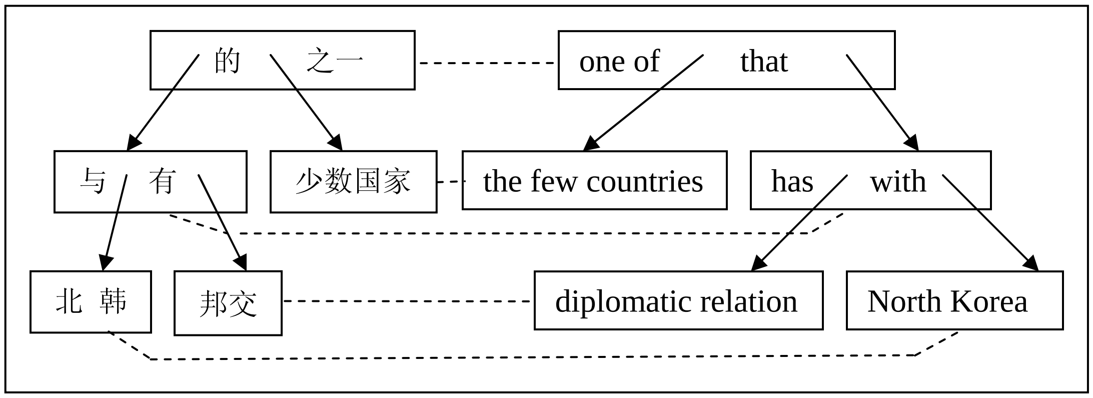


Figure 1-2: an illustration of hierarchical phrase ordering

The hierarchical phrase model calculates $SCORE(E_1^I, F_1^J)$ by recursive structures. It can be seen as one of the recent tree-based SMT models, and further improved the accuracy. More related SMT models are described in Section 2.1.3.

From the above examples, it can be seen that SMT models has evolved towards higher levels of abstraction. The accuracy has been improved by more precise representations of structural correspondence between languages. This fact leads naturally to a question – what abstraction can best represent such correspondence? Possible answers may come from linguistics, as it is the human abstraction of languages.

1.1.3 Grammars – the provider of linguistic information

Being an important part of linguistics, *grammar* studies the rules behind languages. Specifically, the aspect of grammar that does not concern meaning directly is called *syntax*, while the aspects that concern meaning include *semantics* and *pragmatics* (Allen, 1995).

Grammars can be prescriptive (for example "Do not use superlative form when comparing only two objects") or descriptive ("the word 'anything' is used in negative sentences and questions"). Linguists are typically more interested in descriptive grammars. In 1956, Chomsky formalized a generative way to describe grammars. Such grammars regard sentences as the result of recursive symbol generations according to certain rules. In terminology, words in a language are called *terminal symbols*, syntactic information such as part-of-speech (e.g. the noun, the verb etc.; Allen 1995) is called *non-terminal symbols*, and the rules that generate new symbols from existing symbols are called *production rules*. From a special *starting non-terminal* symbol, a sentence can be generated by recursively applying production rules to existing symbols.

Chomsky classified generative grammars into four categories known as the *Chomsky Hierarchy* (Allen, 1995). Among these categories, the *context free grammar* (CFG) is frequently used to represent the syntax of English. In this grammar, every production rule takes a non-terminal symbol and generates a string of new symbols. An example CFG is shown in Figure 1-3. In this grammar, *S, NP, VP, N,* and *V* are non-terminal symbols, representing the start symbol, noun phrases, verb phrases, nouns and verbs, respectively; "I", "like" and "C++" are terminal symbols, which are the vocabulary words in this grammar. This CFG contains six production rules. For example, $VP \rightarrow V\ NP$ is a production rule that generates string $V\ NP$ (verb noun-phrase) from symbol VP (verb-phrase).

S	$\rightarrow$	$NP\ VP$	N	$\rightarrow$	C++
NP	$\rightarrow$	N	VP	$\rightarrow$	$V\ NP$
N	$\rightarrow$	I	V	$\rightarrow$	like

Figure 1-3: an example CFG that consists of six production rules

According to the above grammar, a subset of English sentences can be analysed. For example, Figure 1-4 shows the process in which the sentence "I like C++" is generated. This structural representation is called a *syntax tree*.

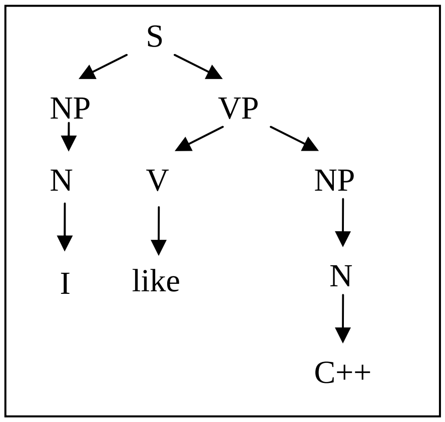


Figure 1-4: an example syntax tree

Because generative grammars have precise description and transformation rules, they are widely used in computer algorithms. Specifically, *parsing* is the process of sentence analysis according to a given grammar. A parsing algorithm is also called a *parser*. For example, a CFG parser can take the input sentence "I like C++" and derive the syntax tree in Figure 1-4, which is also called a *parse tree*. More details of the theory of parsers are reviewed in Section 2.3.

1.1.4 Synchronous grammars and SMT by parsing

Besides English, CFG can also be used to describe Chinese and other languages. Now assume that a CFG in English can have a corresponding CFG in Chinese, in which each symbol and production rule has a counterpart. We can thus combine the two grammars to make a grammar which generates matching Chinese-English sentence pairs simultaneously. Such combined grammars are called *transduction grammars* (Aho and Ullman, 1969), or *synchronous grammars* (Wu, 1997). Figure 1-5 shows a simple synchronous CFG by combining the English CFG in Figure 1-3 and its Chinese counterpart (the non-terminal symbols in the Chinese CFG are underlined).

$\underline{S}\|S$	→	$\underline{NP}\|NP$ $\underline{VP}\|VP$	$\underline{N}\|N$	→	C++|C++
$\underline{NP}\|NP$	→	$\underline{N}\|N$	$\underline{VP}\|VP$	→	$\underline{V}\|V$ $\underline{NP}\|NP$
$\underline{N}\|N$	→	我 | I	$\underline{V}\|V$	→	喜欢 | like

Figure 1-5: a bilingual grammar combining a Chinese and an English grammar

According to the combined grammar in Figure 1-5, Figure 1-6 illustrates the synchronous parse tree for the following sentence pair

$$\begin{bmatrix} \text{我 喜欢 C++} \\ \text{I like C++} \end{bmatrix},$$

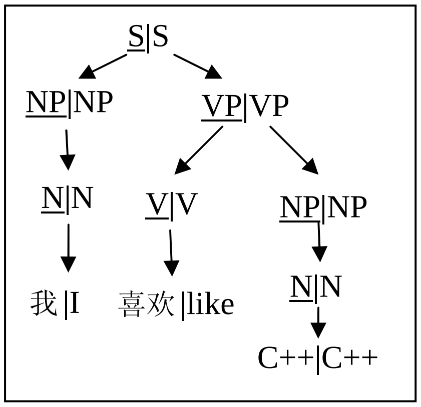


Figure 1-6: an example parse tree for Chinese-English bilingual grammar

Now if a parser can generate the above syntax tree from only one sentence (e.g. "我 喜欢 C++"), then the other sentence (e.g. "I like C++") can be derived from the tree. This is the basic idea of SMT by synchronous parsing – the main theory of this thesis. Details on synchronous grammars are discussed in Chapter 3.

It can be seen that SMT by synchronous parsing is a higher level model than the word-based and phrase-based SMT in Section 1.1.2.2, which use alignment patterns to abstract word and phrase orders. Like hierarchical phrase based SMT, synchronous grammar based SMT use tree structures. However, a potential advantage of SMT by synchronous parsing is the use of syntactic information, which arbitrary hierarchical phrases do not have.

The decoding algorithm for SMT by parsing is different from those models in Section 1.1.2 – the goal of the decoding search is the best synchronous tree, instead of the best translation directly. Therefore, the decoding process can be seen as a statistical parsing process (Section 2.3.2). The advantage is that existing parsing algorithms can be used to facilitate SMT research. For example, the experiments of this thesis are based on GenPar, which is the implementation of a generalised parsing algorithm (Section 2.3.3).

1.1.5 Chinese-English SMT by parsing

The topic of this thesis is Chinese-English SMT by parsing. This is the first application of the SMT by parsing model in Chapter 3 to the specific language pair.

On the one hand, the language-independent mathematical theory of SMT by parsing is important to the effect of translation, and the experimental output reflects the strength of weakness of the SMT model (Chapter 5).

On the other hand, the effect of Chinese-English SMT is also influenced by the specific characteristics of the two languages, and language-specific processing is required for the translation. For example, a Chinese sentence is written as a continuous sequence of characters. To be processed by the SMT systems, it needs to be separated into individual words. This process is called *segmentation* (Jurafsky and Martin, 2000). What is more, the structural correspondence between Chinese and English is more complex than that between Indo-European language pairs. For example, phrase-based models may benefit from the strong localisation effect observed between Indo-European phrases (Tillman et al., 1997), but Chinese phrases may not have the advantage.

This thesis does large-scale experiments with existing software, including the GenPar framework. Apart from the necessary software engineering work for a running system, special engineering techniques such as parallel processing are used to control the experiment progress under the time frame of this thesis. The details of these experiments are recorded in Chapter 4.

The next section gives the organisation of the rest of this thesis.

1.2 Overview

Chapter 2 provides the detailed background, including the mathematical theory of SMT models and MT evaluation methods. By reviewing the theory of parsers, it introduces the generalised parsing algorithm, which is used by GenPar.

Chapter 3 introduces the synchronous grammar used by this thesis, as well as its statistical parsing model and the training and decoding process. On this basis, it introduces the theory of SMT by parsing, showing the implementation of several important algorithms by generalised parsing.

Chapter 4 records the details of the experiments, including the software used, the corpus texts, the process of a typical experiment and brief introductions of the programs written during the process. This chapter also includes a summary of the research questions corresponding to each experiment.

Chapter 5 discusses the research questions with the experiment results. It draws a conclusion of this thesis and gives suggestion for future work.

1.3 Contributions

- Detailed exposition of the theory of SMT and SMT by parsing using the example of Chinese-English translation, giving original analysis to important algorithms.

- First experiments on Chinese-English SMT by parsing using the GenPar framework, with results comparable to existing language pairs using GenPar.

- Demonstration that large-scale Chinese-English syntax-based SMT is possible, by using over 3 million words of bilingual data, 32 machines and 1,000 hours of processing.

- Analysis and comparison of SMT models by novel experiments.

Chapter 2

Background

2.1 The theory of statistical machine translation

As stated in Chapter 1, three important factors for SMT are the training process, the decoding process and the statistical model. Typical SMT models include the *source-channel model* (Brown et al., 1993) and the *log-likelihood model* (Och et al., 2002). This chapter uses the source-channel model for illustration.

2.1.1 The source-channel model

The source-channel model originated from signal processing and information theory, and was first used in statistical natural language processing for speech recognition. It was used for SMT by Brown et al. (1993).

Imagine that there is a noisy channel, through which sentences in one language are distorted into another. Now for the translation problem, the source sentences can be regarded as the result of their corresponding target sentences being passed through such an imaginary channel. The task of MT is to reconstruct the original sentences, given their distorted versions. It is done by searching for a target sentence that has the highest conditional probability given the source sentence.

In mathematical form, the English translation E_1^I (i.e. $e_1,...,e_i,...e_I$) for a given Chinese sentence F_1^J (i.e. $f_1,...,f_j,...f_J$) is the one that satisfies:

$$E_1^I = \arg\max_{E_1^I} P(E_1^I \mid F_1^J) \qquad (2\text{-}1)$$

Equation 2-1 is a specific version of Equation 1-1. This equation uses conditional probability $P(E_1^I \mid F_1^J)$ for the ranking score $SCORE(E_1^I, F_1^J)$.

According to Bayes' rule, the source-channel model estimates $P(E_1^I \mid F_1^J)$ by breaking it into the product $\alpha P(E_1^I)P(F_1^J \mid E_1^I)$ (where α is a normalisation factor). In this equation, the prior probability of the English sentence $P(E_1^I)$ is called the *language model* and the likelihood probability $P(F_1^J \mid E_1^I)$ is called the *translation model*. Interestingly, the Chinese-English translation probability $P(E_1^I \mid F_1^J)$ is now estimated by the English-Chinese translation probability $P(F_1^J \mid E_1^I)$.

The main reason for estimating $P(E_1^I \mid F_1^J)$ by the product $\alpha P(E_1^I)P(F_1^J \mid E_1^I)$ is modularity. While the translation model represents the matching between the source and target sentences, the language model is used specially to control the fluency of output sentence. The implementation of the language model is typically an n-gram model (Brown et al., 1990), while the translation model has evolved from word-based to phrase-based and higher levels of abstractions, as introduced in Section 1.1.2.2.

2.1.2 Word-based SMT – the fundamental ideas

As introduced in Section 1.1.2.2, word-based SMT assumes that sentences are translated word by word, and the different ordering between the input and the translation can be represented by a pattern of word alignment. Take the Chinese-English translation for example. As stated earlier, the translation model estimates the English-Chinese translation probability $P(F_1^J \mid E_1^I)$. Thus it assumes that each English word e_i in a source sentence E_1^I is translated into a Chinese word f_j in its translation F_1^J. Therefore, in equation form, the word alignment between E_1^I and F_1^J can be expressed as $A_1^J = a_1, ..., a_j, ..., a_J$, where a_j stands for the index of the English word that aligns to the Chinese word f_j. Considering alignments, the translation model can be calculated by summing disjoint probabilities:

$$P(F_1^J \mid E_1^I) = \sum_{A_1^J} P(F_1^J, A_1^J \mid E_1^I) \qquad (2\text{-}2)$$

For each alignment A_1^J, the joint probability $P(F_1^J, A_1^J \mid E_1^I)$ can be broken down according to the chain rule of probability, in the order of dependence from J to the sequence of a_1, f_1, …, a_j, f_j, …, until a_J, f_J:

$$P(F_1^J, A_1^J \mid E_1^I) = P(J \mid E_1^I) \prod_{j=1}^{J} P(a_j \mid a_{j-1}, f_{j-1}, J, E_1^I) P(f_j \mid a_j, f_1^{j-1}, J, E_1^I) \qquad (2\text{-}3)$$

Equation 2-3 is still hard to process in both the learning phase and the decoding phase, because there are many factors to consider.

By making independence assumptions over the conditional probabilities, Brown et al. (1993) gave simplifications to equation 2-3 and developed the five seminal word-based IBM models. The simplest model, IBM model-1, can be used as an illustration of the learning and decoding phases in word-based SMT. This model has the following independence assumptions on Equation 2-3:

- $P(J \mid E_1^I)$ = ε (the length J is not considered)
- $P(a_j \mid a_{j-1}, f_{j-1}, J, E_1^I) = \frac{1}{I+1}$ (the alignment of words is evenly distributed) [1]
- $P(f_j \mid a_j, f_1^{j-1}, J, E_1^I) = P(f_j \mid e_{a_j})$ (individual word-to-word translations are independent of each other)

With these independence assumptions, equation 2-3 becomes

$$P(F_1^J, A_1^J \mid E_1^I) = \varepsilon \prod_{j=1}^{J} \frac{P(f_j \mid e_{a_j})}{I+1} = \frac{\varepsilon}{(1+I)^J} \prod_{j=1}^{J} P(f_j \mid e_{a_j}) \qquad (2\text{-}4)$$

Hence the whole translation model of Equation 2-2 becomes

$$P(F_1^J \mid E_1^I) = \sum_{A_1^J} P(F_1^J, A_1^J \mid E_1^I) = \sum_{A_1^J} \frac{\varepsilon}{(1+I)^J} \prod_{j=1}^{J} P(f_j \mid e_{a_j})$$

Further, because words are aligned independently, each word can have I + 1 different alignments:

$$P(F_1^J \mid E_1^I) = \sum_{A_1^J} \frac{\varepsilon}{(I+1)^J} \prod_{j=1}^{J} P(f_j \mid e_{a_j}) = \frac{\varepsilon}{(I+1)^J} \sum_{a_1=0}^{I} \cdots \sum_{a_j=0}^{I} \prod_{j=1}^{J} P(f_j \mid e_{a_j})$$

What is more, by observation, the sum of the permutation production can be rewritten as the production of term sums:

$$P(F_1^J \mid E_1^I) = \frac{\varepsilon}{(I+1)^J} \sum_{a_1=0}^{I} \cdots \sum_{a_j=0}^{I} \prod_{j=1}^{J} P(f_j \mid e_{a_j}) = \frac{\varepsilon}{(I+1)^J} \prod_{j=1}^{J} \sum_{i=0}^{I} P(f_j \mid e_i) \qquad (2\text{-}5)$$

Equation 2-5 is fairly simple to compute, and can be used in the decoding process to determine the most probable translation. The only parameters required in this equation are the word translation probabilities $P(f \mid e)$, which are collected in the learning process and from corpus data.

The ideal situation for learning $P(f \mid e)$ is the availability of a word-aligned corpus. In such a corpus, words that translate to each other are put into pairs, and their frequency recorded. The *maximum likelihood* estimation method (Russell and Norvig, 2003) can be directly applied to get the probabilities $P(f \mid e)$, by choosing the $P(f \mid e)$ that maximises the probability of the corpus data. Suppose

[1] The denominator is I+1 instead of I, because there is a case when a Chinese word is aligned to no English word.

that in the corpus, an English word *e* translates to Chinese words $f_1, ..., f_n, ..., f_N$, each translation occurring $c(f_1 | e)$,…, $c(f_n | e)$,…, $c(f_N | e)$ times, respectively. Assuming the translations are independent, the probability of translation from *e* to f_n occurring $c(f_n |e)$ times among $\sum_{k=1}^{N} c(f_k|e)$ samples is:

$$p = P(f_n | e)^{c(f_n|e)} (1 - P(f_n | e))^{((\sum_{k=1}^{N} c(f_k|e)) - c(f_n|e))} \quad (2\text{-}6)$$

By the maximum likelihood principle, the best value for $P(f_n | e)$ is the one that maximises the probability *p* in Equation 2-6. This value can be derived by finding the equivalent value $P(f_n | e)$ that maximises the logarithm of *p*:

$$\begin{aligned} q &= \log(P(f_n | e)^{c(f_n|e)} (1 - P(f_n | e))^{((\sum_{k=1}^{N} c(f_k|e)) - c(f_n|e))}) \\ &= c(f_n | e) \log P(f_n | e) + ((\sum_{k=1}^{N} c(f_k | e)) - c(f_n | e)) \log(1 - P(f_n | e)) \end{aligned}$$

We then solve the following equation to find the $P(f_n | e)$ that makes the derivative of *q* zero:

$$\frac{dq}{dP(f_n | e)} = \frac{c(f_n | e)}{P(f_n | e)} - \frac{(\sum_{k=1}^{N} c(f_k | e)) - c(f_n | e)}{1 - P(f_n | e)} = 0$$

$$\Rightarrow \qquad P(f_n | e) = \frac{c(f_n | e)}{\sum_{k=1}^{N} c(f_k | e)} \quad (2\text{-}7)$$

The answer $P(f_n | e) = c(f_n | e) / (\sum_{k=1}^{N} c(f_k | e))$ is consistent with intuition – it is simply the number of times that *e* translates to f_n divided by the total number of times that *e* translates to any f_k.

In real world situations, a word-aligned corpus is hard to find. However, it is comparatively easy to find a sentence-aligned corpus, containing pairs of translation sentences. If the word-to-word alignment for these sentences is known, the problem is reduced to the word-aligned corpus learning described above. Alternatively, if the probability distribution of the word alignment for the sentences is known, the word-aligned corpus situation can be approximated by computing the mathematical expectation of word-to-word alignments. For example, given a translation sentence pair (F_1^J, E_1^I) and the alignment distribution $P(A_1^J | F_1^J, E_1^I)$, we can calculate the mathematical expectation of the number of times that *f* in F_1^J is aligned with *e* in E_1^I:

$$\bar{c}(f | e; F_1^J, E_1^I) = \sum_{A_1^J} P(A_1^J | F_1^J, E_1^I) \times (\text{the number of times } f \text{ is aligned to } e \text{ with } A_1^J)$$

Using the Kronecker Delta function[1], this equation can be written as:

$$\bar{c}(f \mid e; F_1^J, E_1^I) = \sum_{A_1^J} P(A_1^J \mid F_1^J, E_1^I) \sum_{j=1}^{J} \delta(f, f_j)\delta(e, e_{a_j}) \tag{2-8}$$

Combining Equation 2-7 and Equation 2-8, we can find an equation to estimate $P(f_n \mid e)$ for the sentence aligned situation:

$$P(f_n \mid e) = \frac{\bar{c}(f_n \mid e)}{\sum_{k=1}^{N} \bar{c}(f_k \mid e)} = \frac{\sum_{A_1^J} P(A_1^J \mid F_1^J, E_1^I)\delta(f, f_j)\delta(e, e_{a_j})}{\sum_{k=1}^{N} \sum_{A_1^J} P(A_1^J \mid F_1^J, E_1^I)\delta(f_k, f_j)\delta(e, e_{a_j})} \tag{2-9}$$

Equation 2-9 cannot be used directly, however, because the alignment distribution $P(A_1^J \mid F_1^J, E_1^I)$ is unknown. One intuitive way in which $P(A_1^J \mid F_1^J, E_1^I)$ can be computed is from the conditional probability rule:

$$P(A_1^J \mid F_1^J, E_1^I) = \frac{P(A_1^J, F_1^J \mid E_1^I)}{P(F_1^J \mid E_1^I)} \tag{2-10}$$

Looking back to Equation 2-4 and Equation 2-5, it can be seen that they give an exact way to compute $P(A_1^J, F_1^J \mid E_1^I)$ and $P(F_1^J \mid E_1^I)$, respectively. Combining Equation 2-4, 2-5 and 2-10, we get:

$$P(A_1^J \mid F_1^J, E_1^I) = \frac{P(A_1^J, F_1^J \mid E_1^I)}{P(F_1^J \mid E_1^I)} = \frac{\prod_{j=1}^{J} P(f_j \mid e_{a_j})}{\prod_{j=1}^{J} \sum_{i=0}^{I} P(f_j \mid e_i)} \tag{2-11}$$

Now we have found a way to compute word-to-word probabilities from alignment probabilities (Equation 2-11), as well as a way to compute alignment probabilities from word-to-word probabilities (Equation 2-9). This is a form of the *expectation maximisation* (EM) algorithm (Russell and Norvig, 2003), with the word alignment probabilities as the hidden parameter. We can find the word-to-word probabilities that bring the corpus probability p (Equation 2-6) to a local maximum, by first assigning initial values to them, and then iterating through Equation 2-11 and 2-9 until the values converge. Generally there are many local maxima for the EM problem. However, for this particular translation model, it can be proved that the local maximum is also the global maximum.

The above method is essentially IBM model-1, although the reasoning is different from the one that Brown et al. (1993) originally used. Models-2 to 5 also derive from Equation 2-3, while they include more factors than model-1 in the calculation. For example, model-2 adds absolute word order to model-1, model-3 further adds fertility (number of words that translates to the word),

[1] Kronecker's delta $\delta(i, j)=1$ when *i* and *j* are the same, and 0 when they are different.

model-4 uses relative word order and model-5 further fixes some deficiencies of model-4.

Before finishing this section, it should be noticed that the word-to-word translations $P(f_n | e)$ (Equation 2-9) and word alignments $P(A_1^J | F_1^J, E_1^I)$ (Equation 2-11) can be used not only for the word-based models, but also as a building block in other SMT models. Extensions on the IBM models have been made for the calculation of word alignment, including the HMM (Russell and Norvig, 2003) model (Vogel et al., 1996) and the bi-directional method (Och et al., 1999). Och and Ney (2003) gives a summary of word alignment methods.

2.1.3 The trend towards higher levels of abstraction

As stated in Section 1.1.2.2, the word-based approach has some apparent problems. For example, English phrases such as "a piece of cake" (which translates to the Chinese word"小菜一碟") or "off the hook" (which translates to the Chinese word "解脱") are not included in the model at all. This is because the alignment model A_1^J provides only one English word index for each Chinese word. As a result, no Chinese word can align to multiple English words.

This problem can be dealt with by phrase-based translation models. Instead of breaking sentences into separate words, these models break sentences into continuous phrases:

$$E_1^I = \tilde{E}_1^K = \tilde{e}_1, ..., \tilde{e}_k ..., \tilde{e}_K \text{, where } \tilde{e}_k = e_{i_{k-1}+1}, ...e_{i_k}$$
$$F_1^J = \tilde{F}_1^K = \tilde{f}_1, ..., \tilde{f}_k ..., \tilde{f}_K \text{, where } \tilde{f}_k = f_{j_{k-1}+1}, ... f_{j_k}$$

Similar to the word-based approach in section 2.1.2, the alignment for phrases from a translation pair can be expressed as $\tilde{A}_1^K = \tilde{a}_1, ...\tilde{a}_k, ...\tilde{a}_K$, where each $\tilde{a}_k$ in $\tilde{A}_1^K$ indicates the English phrase index that aligns to *k*. In other words, alignment $\tilde{A}_1^K$ means that English phrase $\tilde{e}_{\tilde{a}_k}$ translates to Chinese phrase $\tilde{f}_k$.

In the decoding process, the translation probabilities can be calculated by summing up possible phrase alignments:

$$P(\tilde{F}_1^K | \tilde{E}_1^K) = \sum_{\tilde{A}_1^K} P(\tilde{F}_1^K, \tilde{A}_1^K | \tilde{E}_1^K)$$

It should be noticed that this equation is similar to Equation 2-2. By the same reasoning from section 2.1.2, the following generalised score function can be derived for phrases:

$$P(\tilde{F}_1^K \tilde{A}_1^K \mid \tilde{E}_1^K) = \prod_{j=1}^{K} P(\tilde{a}_k \mid \tilde{a}_{k-1}, \tilde{f}_{k-1}, \tilde{E}_1^K) P(\tilde{f}_k \mid \tilde{a}_k, \tilde{f}_1^{k-1}, \tilde{E}_1^K)$$

Most current phrase-based methods are simplifications of this equation. For example, Koehn et al. (2003) used the following simplifications in their summary of phrase-based approaches:

- $P(\tilde{F}_1^K \mid \tilde{E}_1^K) = \arg\max_{\tilde{A}_1^K} P(\tilde{F}_1^K, \tilde{A}_1^K \mid \tilde{E}_1^K)$ (replace summation by maximum)
- $P(\tilde{a}_k \mid \tilde{a}_{k-1}, \tilde{f}_{k-1}, \tilde{E}_1^K) = P(j_{k-1} \mid i_{k-1})$ (the phrase alignment is only dependent on corresponding phrase positions)
  or $P(\tilde{a}_k \mid \tilde{a}_{k-1}, \tilde{f}_{k-1}, \tilde{E}_1^K) = P(j_{k-1} \mid i_{k-1}) = d(j_{k-1} + 1 - i_{k-1}) = \alpha^{|j_{k-1}+1-i_{k-1}|}$ (still further simplification – the phrase alignment is only dependent on the distance between corresponding indice)
- $P(\tilde{f}_k \mid \tilde{a}_k, \tilde{f}_1^{k-1}, \tilde{E}_1^K) = P(\tilde{f}_k \mid \tilde{a}_k)$ (the individual phrase-to-phrase translations are independent of each other)

There are alternative ways to train a phrase-based system. Firstly, given a word alignment method (Section 2.1.2), phrase alignment can be derived from word-to-word alignment (Och et al., 1999, Koehn et al., 2003) – two phrases are aligned when all the words in one phrase are only aligned to words in the other. Or alternatively, phrase alignment can also be derived directly by EM machine learning methods similar to the word alignment from Section 2.1.2. Marcu and Wong (2002) used a (simplified) method similar to IBM model-3 to compute phrase alignments directly.

The phrase-based models are able to solve the one-to-many word-alignment problem successfully. However, they still assume that sentences are translated by simple reordering. As shown in Section 1.1.2.2, this assumption is oversimplified for complex situations. Consequently, several tree-based models are proposed for better solutions. An example is the hierarchical phrase based model (Chiang, 2005) in 1.1.2.2. This model does not take use of syntactic information. The training process derives hierarchical phrase information from existing statistical word-to-word alignments. In comparison, Yamada and Knight (2001) proposed a model that uses the syntactical information of the input language. This model assumes that the translation is derived by operations over the CFG parse tree of the input sentence. It takes use of a CFG parser, and can be seen as a tree-to-string model. Meanwhile, there are also tree-to-tree models which take use of syntactic information of both the input and the translation. For example, the work of Gildea (2003) includes such models. Cowan et al. (2006) proposed a tree-to-tree model which maps the parse tree of the input language to the parse tree of the translation language.

As stated in Section 1.1.4, SMT by synchronous parsing can be seen as a specific syntax tree based SMT model. This model is based on combined syntax trees. Its decoding algorithm can benefit from existing parsing algorithms. Examples of SMT by synchronous parsing include research by Wu (1997), as well as Melamed and Wang (2005). By using the GenPar framework, this thesis uses the model of Melamed and Wang (2005). The detail of the synchronous grammar will be given in Chapter 3.

2.2 The evaluation of machine translation

It is important to evaluate the accuracy of machine translation against fixed standards, so that the effect of different models can be seen and compared. The obvious difficulty in setting a standard for MT evaluation is the flexibility of natural language usage. For an input sentence, there can be many perfect translations. Knight and Marcu (2004) showed 12 independent English translations by human translators, given the same Chinese sentence. All of the 12 are different, yet all correct.

The most accurate evaluation is human evaluation, and it is frequently used for new MT theories. However, this method is far more time consuming than automatic methods. It is difficult for human evaluators to evaluate a large sample of translated sentences. Research has shown that certain machine evaluation methods correspond reasonably well with human evaluators, and thus they are usually used for the evaluation of large test sets. This section introduces three most common automatic evaluation methods, which are used by the experiments of this thesis.

2.2.1 The Bleu metrics

The Bleu metrics (Papineni et al., 2001) evaluates machine translation by comparing the output of an MT system with correct translations. Therefore, a test corpus is needed for this method, giving at least one manual translation for each test sentence. During a test, each test sentence is passed to the MT system, and the output is scored by comparison with the correct translations. This score is called the *Bleu score*. The output sentence is called the *candidate* sentence, and the correct translations are called *references*.

The Bleu score is evaluated by two factors, concerning the precision and the length of candidates, respectively. *Precision* refers to the percentage of correct n-grams in the candidate. In the simplest case, unigram (n=1) precision equals to the number of words from the candidate that appear in the references divided by the total number of words in the candidate.

The standard n-gram precision is sometimes inaccurate in measuring translation accuracy. Take the following candidate translation for example:

Candidate: *a a a.*

Reference: *a good example.*

In the above case, the standard unigram precision is 3/3=1, but the candidate translation is inaccurate with duplicated words. Because of this problem, Bleu uses a *modified n-gram precision* measure, which consumes a word in the references when it is matched to a candidate word. The modified unigram precision of the above example is 1/3, for the word 'a' in the reference is consumed by the first 'a' in the candidate.

Similar to unigrams, modified n-gram precision applies to bigrams, trigrams and so forth. In mathematical form, the n-gram precision is as follows:

$$p_n = \frac{\sum_{c \in \{Candidate\}} \sum_{n\text{-}gram \in C} Matched(n\text{-}gram)}{\sum_{c \in \{Candidate\}} \sum_{n\text{-}gram \in C} Count(n\text{-}gram)}$$

Apart from modified n-gram precision, a factor of candidate length is also included in the Bleu score. The main aim of this factor is to penalise short candidates, because long candidates will be penalised by low modified n-gram precisions. Take the following candidate for example:

Candidate: *C++ runs.*

Reference: *C++ runs much faster than Python.*

Both the unigram precision and the bigram precision for the above candidate are 1 (i.e. 100%), but the candidate contains much less information than the reference. To penalise such short candidates, a *brevity penalty* score is used. Suppose that the length of the reference sentence is r, and the length of the candidate is c. In equation form, the brevity penalty score is as follows:

$$BP = \begin{cases} 1 & \text{if } c > r \\ e^{(1-r/c)} & \text{if c} \leq \text{r} \end{cases}$$

When there are many references, *r* takes the length of the reference that is the closest to the length of the candidate. This length is called the *effective reference* length.

The Bleu score combines the modified n-gram score and the brevity penalty score. When there are many test sentences in the test set, one Bleu score is calculated for all candidate translations. This is done is two steps. Firstly, the geometric average of the modified *n*-gram precisions p_n is calculated for all *n* from 1 to *N*, using positive weights w_n which sum up to 1. Secondly, the brevity penalty score is computed with the total length of all candidates and total effective reference length for all candidates. In equation form,

$$B_{LEU} = BP \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n \right)$$

By default, the Bleu score includes the unigram, bigram, trigram and 4-gram precisions, each having the same weight. This is done by using N=4 and w_n=1/N in the above equation.

Experiments have shown that the Blue metrics are generally consistent with human evaluators, and thus are useful indicators for the accuracy of machine translation.

2.2.2 The NIST metric

The NIST metric (Doddington, 2002) was developed on the basis of the Bleu metrics. It focuses mainly on improving two problems of the Bleu score. Firstly, the Bleu metrics use the geometric average of modified n-gram precisions. However, because current MT systems have not reached considerable fluency, the modified *n*-gram precision scores may become very small for long phrases (i.e. big *n*). Such small scores have a potential negative effect on the overall score, which is not desired. To solve this problem, the NIST score uses the arithmetic average instead of geometric average. In this way, all modified n-gram precisions make zero or positive contribution to the overall score.

Secondly, the Bleu metrics weigh all n-grams equally in the modified n-gram precision score. However, some n-grams carry more useful information than others. For example, the bigram “washing machine” is considered more useful for the evaluation than the bigram “of the”. The NIST metric gives each n-gram an information weight, which is computed by:

$$Info(w_1 \ldots w_n) = \log_2 \left(\frac{\text{the \# of occurrences of } w_1 \ldots w_{n-1}}{\text{the \# of occurrences of } w_1 \ldots w_n} \right)$$

Besides the above two differences, the NIST score also uses a special brevity penalty score. In equation form, it can be written as:

$$BP = \exp\left(\beta \log^2 (\min(\frac{L_{sys}}{\overline{L}_{ref}}, 1)) \right),$$

where $\overline{L}_{ref}$ is the average number of words in the references, L_{sys} is the number of words in the candidate, and β is chosen to make BP=0.5 when the number of words in the candidate is 2/3 of the average number of words in the references.

In summary, the NIST score for MT evaluation can be written as:

$$Score = BP \cdot \sum_{n=1}^{N} \left(\frac{\sum_{w_1 \ldots w_n \in Matched} Info(w_1 \ldots w_n)}{\sum_{w1 \ldots wn \in Candidate} (1)} \right)$$

2.2.3 The F-measure

The F-measure (Turian et al., 2003) is an MT evaluation method developed independently from the Bleu and NIST metrics. In the domain of natural language processing, the term *F-measure* refers to a combination of *precision* and *recall*. It is commonly used for the evaluation of information retrieval systems. Suppose that the set of candidates is *Y* and the set of references is *X*, the precision, recall and F-measure are defined as follows:

$$precision(Y \mid X) = \frac{|X \cap Y|}{|Y|}$$

$$recall(Y \mid X) = \frac{|X \cap Y|}{|Y|}$$

$$F\text{-}measure = \frac{2 \times precision \times recall}{precision + recall}$$

In the simplest case, the F-measure for a MT translation candidate can be based on unigram precision and recall. See Figure 2-1 for an illustration of this method.

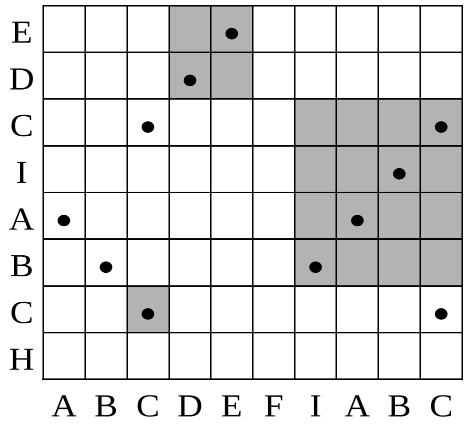


Figure 2-1: unigram matches; adapted from (Turian et al., 2003).

In the above figure, each row represents a unigram (i.e. word) from the candidate translation (*C*), and each column represents a unigram from a reference (*R*). A dot (●) highlights the matching between a row and a column, which is called a *hit*. A *matching* is a subset of hits in which no two are in the same row or column. For the unigram case, the *size* of a matching can be defined as the number of hits in it. A matching with the biggest size is called a *maximum matching*, and is used as $R \cap C$ for precision and recall computations. Figure 2-1 shows a maximum matching with dark background.

Denote the size of a maximum matching as *MMS*. In equation form, we have:

$$precision(C \mid R) = \frac{|MMS(C,R)|}{|C|}$$

$$recall(C \mid R) = \frac{|MMS(C,R)|}{|R|}$$

Therefore, from the above definitions, the unigram F-measure can be calculated.

The unigram form of the F-measure treats each sentence as a bag of words. This method ignores the evaluation of the word order in the candidate translations. One way to include the word order information is weighing continuous hits (i.e. phrases) more heavily than discontinuous hits. In formal definition, a *run* is a sequence of hits in which both the row and the column are contiguous. For example, the matching in Figure 2-1 contains three runs, each with length 1, 2 and 4 respectively. Denote a matching with *M*, and a run in *M* with *r*. To give longer runs more weight, the size of matching *M* can be calculated by:

$$size(M) = \sqrt[e]{\sum_{r \in M} length(r)^e} \tag{2-12}$$

In the above equation, *e* is the weighing factor which favours longer runs when *e*>1. When *e*=1, the F-measure is reduced to the unigram case.

2.2.4 Summary

Experiments have shown that automatic evaluation methods are useful indicators of the quality of MT. However, they are not always consistent with human evaluators. Also, among different evaluation methods, some may perform comparatively better in certain cases but worse in others. For example, with the reference “programming methods”, the candidate “methods of programming” would have a comparatively low Bleu score, because it does not contain matching bigrams. The same candidate may have a better score by the unigram F-measure, because word order information is not considered by this method. Therefore, the unigram F-measure is more consistent with human evaluators in this particular example. In contrast, the candidate “methods programming of” will not be penalised by the unigram F-measure by the same reason. Therefore, the Bleu metrics will be more consistent with human evaluators in this case.

The three automatic methods from Section 2.2.1 to Section 2.2.3 are currently the most commonly used for MT evaluation. In the experiments of this thesis, all three methods are applied.

2.3 The theory of parsing and generalised parsing

As stated in Section 1.1.3, *parsing* refers to the process of sentence analysis in order to decide its grammatical structure. It is not only a means to bring syntactic information to SMT, but also an essential algorithm used by the synchronous parsing based SMT model. By reviewing important parsing algorithms, this section introduces the generalised parsing algorithm that GenPar implements.

Similar to the development of MT algorithms (Section 1.1.1), natural language parsing algorithms have also evolved from rule-based to statistical approaches. Section 2.3.1 and Section 2.3.2 reviews these two approaches, respectively, giving original analysis to some important algorithms.

2.3.1 Rule-based parsers

According to a grammar, rule-based parsers search for the best parse tree by hard-coded rules. There are generally two searching directions: *top-down parsing*, which starts from the starting non-terminal and generates possible partial parse trees to match the input sentence, and *bottom-up parsing*, which starts from the input sentence and hypothesise possible partial parse trees to reach the starting non-terminal backwardly.

Because production rules are applied recursively, different parse trees may share the same sub parse trees. To avoid these overlapping sub parse trees being produced more than once, the parser can cache them (i.e. memoisation) during the parsing process. Hence the parse problem can be efficiently solved by dynamic programming (DP) (Cormen et al., 2001). In DP-based parsers, the cache for sub parse trees is called the *chart*. Consequently, the DP-based parsers are called *chart parsers*.

The CYK (or CKY) algorithm and the Earley (1970) algorithm are two examples of rule-based chart parsers for CFG. The CYK algorithm was developed by Cocke (1970), Kasami (1965) and Younger (1967). It is a pure bottom-up DP-based search algorithm, which caches the partial parse trees of all possible subsequences of an input sentence. The CYK algorithm has a restriction on the CFG – it requires that each production rule to be in the form of either $A \rightarrow BC$ or $A \rightarrow \alpha$, where *A*, *B* and *C* are non-terminal symbols (*B*, *C* are not the starting non-terminal symbol) and α is a terminal symbol. Such restricted form of CFG is called the *Chomsky Normal Form* (CNF). Under this restriction, the time complexity of the parser can be largely reduced.

Suppose that the input sentence is a_1, …, a_n, and the grammar contains non-terminal symbols R_1, …, R_r. Further, suppose that that the start symbol is R_{start}. Because the grammar is in CNF, each rule can be written either as $R_x \rightarrow a_y$ or as $R_x \rightarrow R_y R_z$. The CYK algorithm contains two bottom-up actions: "*scan*", which builds partial parse trees from a terminal symbol *a* with a production rule $R_x \rightarrow a$; and "*compose*", which builds partial parse trees from two non-terminals R_y and R_z with a production rule $R_x \rightarrow R_y R_z$. The cache (memoisation table) for the CYK algorithm can be represented by an *n* by *n* by *r* table *Chart*, where *Chart*[*start*, *end*, *symbol*] caches whether the subsequence (of input sentence) starting from index *start* and ending at index *end* can be generated from *symbol*. Figure 2-2 illustrates the algorithm.

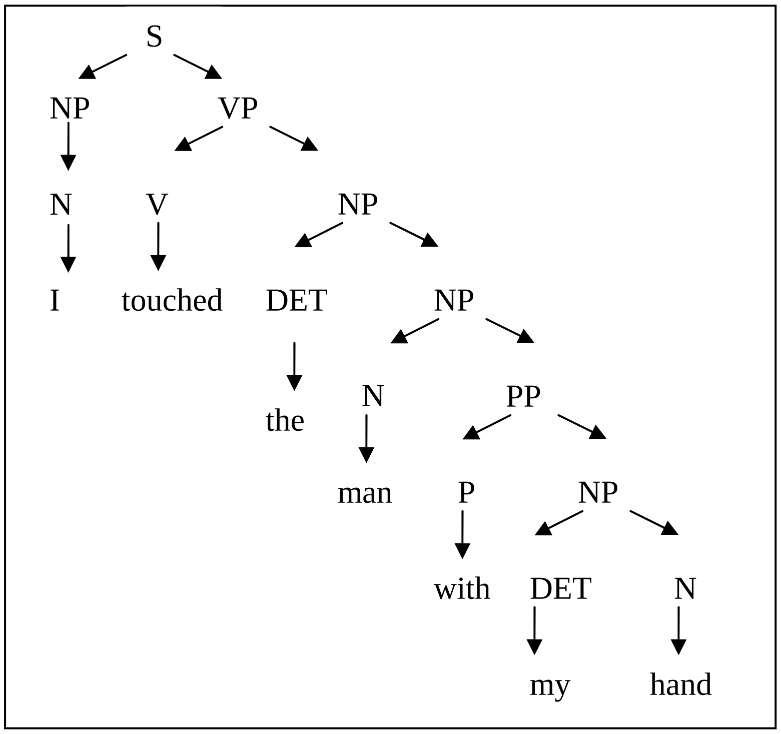
```
procedure Scan(index):
   for each rule $R_x$ -> $a_{index}$:
      Chart[index,index,$R_x$] = True


procedure Compose(start, end, middle):
   for each production $R_x$ -> $R_y$ $R_z$:
      if Chart[start,middle,$R_y$] == True
         and Chart[middle+1,end,$R_z$]:
         Chart[start,end,$R_x$] = True

function CYK():
   initialise Chart with each entry = False
   for each index in [1..n]:
      Scan(index)
   for each length in [2..n]:
      for each start in [1..n-length+1]:
         for each middle = [start..start+length-1]:
            Compose(start,start+length-1,middle)

   return Chart[1,n,$R_{start}$]
```

Figure 2-2: an illustration of the CYK algorithm

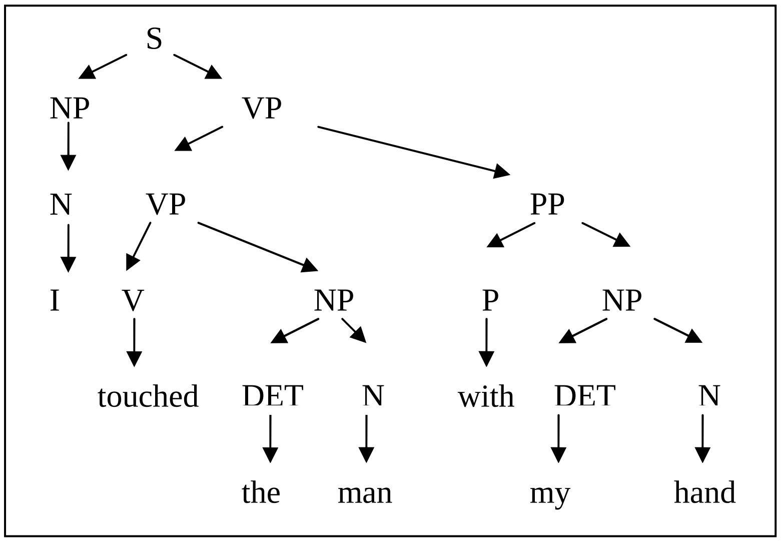
Compared to the CYK algorithm, the Earley algorithm is more complex. It combines bottom-up search with top-down prediction. The algorithm processes the input sentences from left to right, and its chart contains the partial parse trees that cover the processed subsequences from the beginning of the sentence. It contains three actions: “*predict*”, which predicts all possible non-terminals that are consistent with the left-most unprocessed word, “*scan*”, which matches the left-most unprocessed word to a rule that generates it, and “*complete*”, which finishes predictions and moves the current word index to the right. Unlike the CYK algorithm, the Earley algorithm does not require its CFG to be in CNF. See (Jurafsky and Martin, 2000) for the details of this algorithm.

In the area of programming languages, rule based parsers have been used for the task of translation. For example, most compilers do the job of translating one formal language to another. However, translation is much harder in the domain of natural language. Ambiguity of structure appears to be the most important obstacle. For example, the sentence “I touched the man with my hand.” can be generated in both ways shown in Figure 2-3. However, only the syntax in Figure

2-3 (b) has the obvious meaning.



(a) This is correct in syntax, but its meaning is not obvious



(b) This is correct, and has the most likely meaning

Figure 2-3: ambiguous sentence

In natural languages, such ambiguities are easy to find. Because there are no simple rules to cover the general case, it is hard for rule based parsing to resolve such ambiguities. In contrast, statistical parsing algorithms collect statistical data from correctly parsed sentences, and resolves ambiguity by experience. Because of this, it has the potential to generalise better to new situations.

2.3.2 Statistical parsers

The idea of statistical parsers originated in the late 1980s, when people started to investigate corpus based English grammar. This is mainly because rule-based grammars never covered the whole usage of the language, and it was doubtful whether they would ever do so. Pioneered by Geoffrey Leech and Roger Garside, linguists started to look for ways to define English grammar from examples – using human annotated corpora (Garside et al., 1987). Meanwhile, study on statistical parsing began.

Similar to rule-based parsers, the statistical parsing process can be viewed as a search algorithm. However, instead of using rules to find the correct parse tree, statistical parsers select the best parse tree from possible candidates. Similar to SMT, the statistical parsing process is called a *decoding* process. In equation form, the decoding process finds the parse tree *T* for the sentence *S* that satisfies:

$$T = \arg\max_T P(T \mid S) \qquad \text{(2-13)}$$

Similar to SMT, the detailed method to compute scores $P(T \mid S)$ is defined by a statistical *model*. The difference between a grammar and a model is worth noticing. A grammar defines what parse trees are allowed, and it is not confined to the scope of statistical parsers. Meanwhile, a model is the method by which candidates are evaluated in statistical parsing.

One of the earliest works in statistical parsing was Sampson (1986). It used a manually designed language model based on a set of transition networks, and the stimulated annealing decoding search algorithm (Russell and Norvig, 2003).

Regarding statistical parsing with CFG, an important early model is the *Probabilistic Context Free Grammar* (PCFG), which associates each production rule with a probability. With PCFG, a candidate parse tree can be scored by the overall probability of the rules used to generate it. Suppose each rule in a parse tree is $LHS_i \rightarrow RHS_i$, then the probability of the parse tree is:

$$P(T \mid S) = \prod_i P(RHS_i \mid LHS_i) \qquad \text{(2-14)}$$

In a similar way to CFG, PCFG can also be parsed with a chart-based DP algorithm. Figure 2-4 illustrates this algorithm.

```
procedure Scan(index):
   for each rule $R_x$ → $a_{index}$:
      Chart[index,index,$R_x$] = P($a_{index}$|$R_x$)


procedure Compose(start, end, middle):
   for each production $R_x$ -> $R_y$ $R_z$:
      Chart[start,end,$R_x$] =
               max( Chart[start,end,$R_x$],
                    Chart[start,middle,$R_y$] *
                    Chart[middle+1,end,$R_z$] * P($R_y$ $R_z$|$R_x$)
                   )

function Viterbi($R_{start}$):
   initialise Chart with each entry = 0
   for each index in [1..n]:
      Scan(index)
   for each length in [2..n]:
      for each start in [1..n-length+1]:
         for each middle = [start..start+length-1]:
            Compose(start,start+length-1,middle)

   return Chart[1, n, $R_{start}$]
```

Figure 2-4: an illustration of the chart parsing algorithm for PCFG (written in the same format as CYK in Figure 2-2)

Figure 2-4 uses the same symbols as Figure 2-2. However, in contrast to the CYK algorithm, *Chart*[*start*, *end*, *symbol*] for the PCFG parsing algorithm records the score for *symbol* to generate the fraction of input starting from index *start* to index *end*, rather than Boolean values. Correspondingly, the function *Compose* computes the new probability instead of the truth value.

It can be seen from Figure 2-4 that according to the PCFG model, the score of a parse tree is produced recursively by the factors $P(RHS \mid LHS)$. Similar to SMT, these production rule probabilities can be derived by a *training* process.

Training can be conducted under two different conditions. Firstly, maximum likelihood learning (Section 2.1.2) can be used when a corpus is available. Such a corpus containing manually parsed sentences is called a *treebank*. Suppose that in a treebank there are 1000 production instances starting with symbol *VP*, among which 600 are $VP \rightarrow V\ NP$. According to maximum likelihood estimation, the probability $P(V\ NP \mid VP)$ is the one that maximises the likelihood of the treebank, which is 600 / 1000 = 0.6. In the same way, all production probabilities in the model can be trained from the treebank.

Secondly, when treebanks are not available, training can be conducted with an EM machine learning algorithm (Section 2.1.2) that uses hidden variables. This algorithm trains with a set of grammatically correct sentences, and takes the parse trees for these sentences as the hidden variables. Specifically, we have the following two conditions: (1) When the parse trees for the training sentences are known, the set of training sentences becomes a treebank, and thus the production rule probabilities $P(RHS \mid LHS)$ can be derived using the maximum likelihood training method above. (2) Meanwhile, when $P(RHS \mid LHS)$ are given, the parse trees can be derived by a parsing algorithm like Figure 2-4. Therefore, with some initial values of $P(RHS \mid LHS)$, the probabilities can be estimated by several EM iterations between (1) and (2), which will find a local maximum of the probability of the training set.

An example of the EM training algorithm is the Inside-Outside algorithm (Baker, 1979) (Lari and Young, 1990). It works differently from the above analysis, but it can be seen essentially as a variation of the above EM algorithm. We explain the difference in the following way.

In an EM iteration, the Inside-Outside algorithm does not actually derive parse trees as the above step (2) does. Instead, for the maximum likelihood estimation of $P(RHS \mid LHS)$ in step (1), it counts the "occurrences" of $LHS \rightarrow RHS$ by the probability of it being used in parse trees. To achieve this, it defines two types of probabilities. Given an input sentence a_1, …, a_n, the *inside probability* for a non-terminal symbol R and a subsequence a_i, …, a_j (written as $Inside(R, i, j)$) is the probability of R generating subsequence a_i, …, a_j, regardless of the intermediate steps; while the *outside probability* (written as $Outside(R, i, j)$) is the probability of R_{start} generating a_1, …, a_{i-1}, a_{j+1}, …, a_n, and R, regardless of the intermediate steps. These probabilities are illustrated in Figure 2-5.

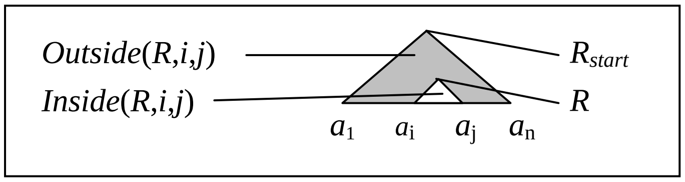


Figure 2-5: an illustration of the inside and outside scores

The above figure illustrates the full parse trees for sentence a_1, …, a_n, as well as *Inside*(R,i,j) (white) and *Outside*(R,i,j) (grey). In these parse trees, the probability of the production rule $R \rightarrow R_x R_y$ being used is:

$$\frac{\sum_k Inside(R_x, i, k) \times Inside(R_y, k, j) \times Outside(R, i, j) \times P(R_x R_y \mid R)}{Inside(R_{start}, 1, n)}$$

During the maximum likelihood estimation of the probabilities $P(RHS \mid LHS)$, the above expression can be used as the count of $R \rightarrow R_x R_y$'s "occurrences" in parse trees. In this way, the above step (1) can be achieved.

Correspondingly, the above step (2) needs to compute the Inside and Outside probabilities from the production rule probabilities $P(RHS \mid LHS)$. Take the inside probability for example, it can be computed as the sum of all partial parse trees from R to a_i, …, a_j, as illustrated in Figure 2-6.

```
procedure Scan(index):
   for each rule $R_x$ → $a_{index}$:
      Chart[index,index,$R_x$] = P($a_{index}$|$R_x$)


procedure Compose(start, end, middle):
   for each production $R_x$ -> $R_y$ $R_z$:
      Chart[start,end,$R_x$] = Chart[start,end,$R_x$] +
                             Chart[start,middle,$R_y$] *
                             Chart[middle+1,end,$R_z$] *
                             P($R_y$ $R_z$|$R_x$)

function Inside(R,i,j):
   initialise Chart with each entry = 0
   for each index in [i..j]:
      Scan(index)
   for each length in [i+1..j]:
      for each start in [i..j-length+1]:
         for each middle = [start..start+length-1]:
            Compose(start,start+length-1,middle)

   return Chart[i, j, R]
```

Figure 2-6: the Inside algorithm for PCFG learning

In the above way, the variation of EM iteration steps (1) and (2) are achieved.

PCFG parsing resolves the ambiguity in Figure 2-3, because Figure 2-3 (b) will be chosen without ambiguity when it is scored higher than Figure 2-3 (a). Meanwhile, PCFG has an obvious disadvantage in that the rules are assigned with probabilities without context. Because of this, PCFG will not be able to resolve the ambiguity between “I touched the man with my hand” and “I saw the man with my wallet”.

Magerman (1995) developed a decision tree (Russell and Norvig, 2003) based statistical parser that includes context information. A parse tree in Magerman’s system includes not only production rules, but also head words and the part of speech of the head words. The *head word* (Allen, 1995) of a phrase is the word that carries the essential meaning of the phrase. For example, the head word of “a big apple” is “apple”, and the head word of “run fast” is “run”. By using head words and their POS, Magerman takes context information into account. Magerman’s parser uses a decision tree (Russell and Norvig, 2003) as the statistical model. The decision tree is trained with a treebank, using information about the current node as well as its four neighbours, including the productions, the head words, the POS of head words and the position information. For the decoding search, Magerman uses a combination of stack decoding (Jelinek, 1969), which is essentially the same as the A* search (Russell and Norvig, 2003) algorithm, and breadth-first search (Russell and Norvig, 2003) with pruning. Magerman’s parser greatly improved the accuracy of CFG parsing.

After Magerman’s decision tree parser, Collins (1996) developed a dependency based statistical CFG parser. Collins’ parser was much simpler than Magerman’s, and yet gave comparable accuracy. Context information was introduced into Collins’ parser by dependency structures, which represent the relationship between a word in a phrase and the head word of the phrase. For example, “big” is dependent of “apple” in “a big apple”, and “fast” is dependent on “run” in “run fast”. Collins’ parser transfers a parse tree into the corresponding dependencies, which are evaluated instead of the tree. For example, the corresponding dependencies for Figure 1-4 are shown in Figure 2-7.

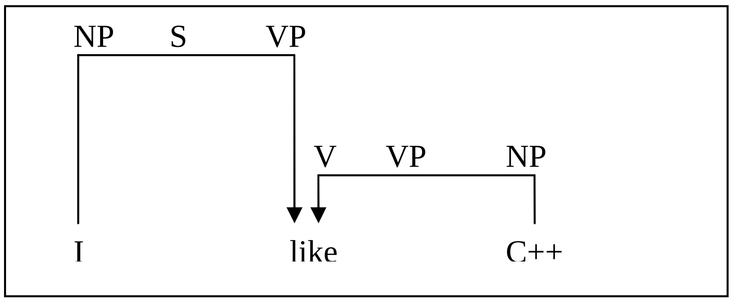


Figure 2-7: the corresponding dependency structure of Figure 1-4

Before the dependencies are evaluated, Collins' parser reduces non-recursive noun phrases (called *baseNPs*) in the input sentence into their head words. For example, "a big apple" will be reduced to "apple" before parsing. This helps to improve the accuracy of dependency structures by removing dependency within noun phrases. With basedNPs considered, Collins' parser works in two steps. Firstly identify the baseNPs *B* from the sentence, then analyse the dependencies *D* on the basis of *B*. In equation form, *T*=(*B*,*D*) and

$$P(T \mid S) = P(B, D \mid S) = P(B \mid S)P(D \mid B, S)$$

With a treebank, the probabilities for the dependencies can be trained with the relative frequencies. Thus Collins' parser is comparatively very quick to train. Collins' parser uses a simple bottom-up chart parsing algorithm, and has reasonable efficiency.

Collins (1997) further improved the parser by proposing three different statistical models. Other statistical models for lexicalised PCFG include the maximum entropy based model (Charniak, 2000). With re-ranking and other techniques, the model reaches the best accuracy in the literature (McClosky et al., 2006).

2.3.3 Generalised parsing

Like most AI algorithms, parsing is essentially a search problem. In this sense, both rule based parsing and statistical parsing are under the same framework. Summarising similarities among parsers, Goodman (1998) outlined a *semiring parsing algorithm* which summarises a wide range of problems including both rule based parsing and probabilistic parsing.

A semiring (Rosenfeld, 1968) is an algebraic set that is closed under two operators ⊕ and ⊗, and that conform to a set of restrictions, such as the identity elements 0 and 1, and the distribution of ⊗ over ⊕. For example, non-negative integers make a semiring, with numerical addition being ⊕ and numerical multiplication being ⊗. 0 and 1 are the identity elements for ⊕ and ⊗, respectively. Boolean values also make a semiring, with AND being ⊕ and OR being ⊗. False and True are the identity elements for ⊕ and ⊗, respectively.

The same concept of semiring can be used to describe different mathematical sets. Goodman (1998) utilised this idea, and used the same general parsing algorithm to describe different specific parsing algorithms. The basis of semiring parsing is the similarity between parsers. For example, Figure 2-2, Figure 2-4 and Figure 2-6, as written in similar structures, shows the similarity between the three algorithms. As Goodman pointed out, they can be viewed as the same generalised parsing algorithm under different semirings. For a specific parser, the semiring defines the specific method for scoring candidates in the search algorithm. Observing the *Compose* function, it can be seen that the semiring for the CYK algorithm has the Boolean set, with ($\wedge$, $\vee$) as the operators ($\oplus$,$\otimes$). Meanwhile, the semirings for the Viterbi and Inside algorithms both have the non-negative floating point set, while they have (+, max) and (+, $\times$) as ($\oplus$, $\otimes$), respectively.

To put such generalisation into practical efficiency, Melamed (2005) suggested a generalised parsing algorithm that is based on semiring parsing. It consists of five components: a *grammar*, a *logic*, a *semiring*, a *search strategy* and a *termination condition*. As the name suggests, grammar defines terminal and non-terminal symbols, as well as a set of production rules. Logic defines the mechanism how the parser runs by generating new partial parse trees. The semiring defines how partial parse trees are scored, the search strategy defines the order in which partial parse trees are processed, and the termination condition defines when to stop the logic necessarily.

Using G for the grammar, L for the logic, R for the semiring ($\oplus_R$ and $\otimes_R$ for the operators and 0_R and 1_R for the identity elements), S for the search strategy and C for the termination condition, the generalised parsing algorithm can be illustrated in brief as Figure 2-8.

```
initialise the R value for symbols and production rules from G
while C isn't met:
   get a set of symbols X by S
   pass X to L to generate a set of symbols Y
   assign R value for symbols in Y
```

Figure 2-8: generalised parsing

With one or many of the five components specifically defined, the generalised parser can be turned into a specific parser such as the CYK parser. In this way the generalised parser saves development time by module reusing. Starting with the synchronous grammar for the experiments, the next chapter shows how this generalised parsing algorithm can be used to facilitate the implementation of various algorithms in SMT by parsing.

Chapter 3

The theory of SMT by parsing

3.1 Statistical GMTG and its generalised parsing

The concept of SMT by parsing was introduced in Section 1.1.4, using a simple synchronous CFG. Moving from this oversimplified example, this section derives the *generalised multitext grammar* (GMTG; Melamed et al., 2004), which is able to express more general parallel structures, and is used in the experiments. Then it introduces the probabilistic version of this grammar, and the corresponding statistical learning and decoding (parsing) algorithms with generalised parsing.

3.1.1 Generalised multitext grammar (GMTG)

3.1.1.1 Reordering

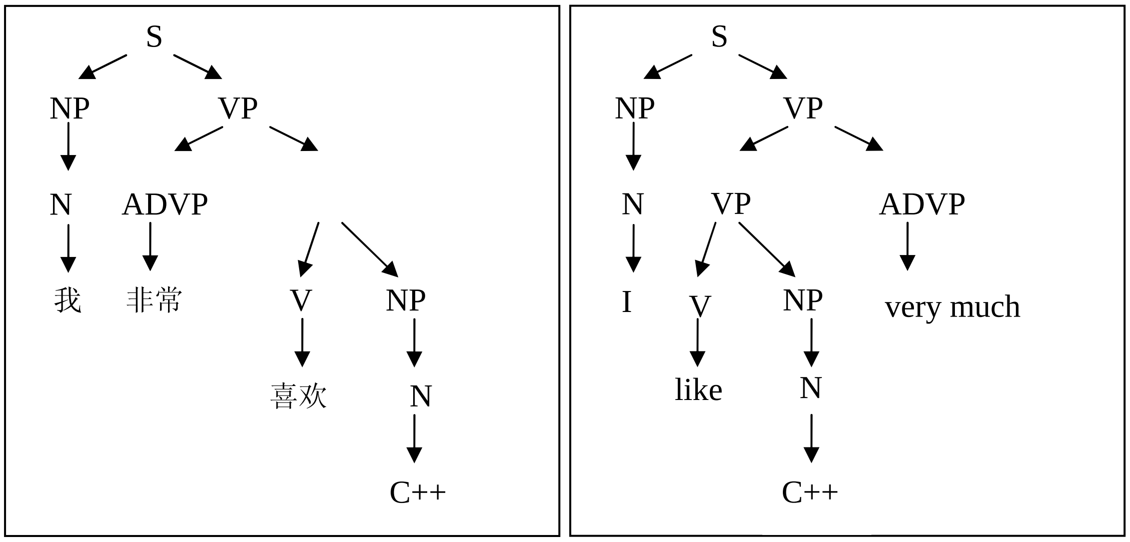
The grammar in Figure 1-5 combines an English CFG with its exact counterpart in Chinese. It can be used to analyse the following sentence pair:

$$\begin{bmatrix}\text{我 喜欢 C++}\\ \text{I like C++}\end{bmatrix}$$

However, the same type of grammar is unable to describe sentence pairs in different word order. Take the following sentence pair for example:

$$\begin{bmatrix}\text{我 非常 喜欢 C++}\\ \text{I like C++ very much}\end{bmatrix}$$

As illustrated in Figure 3-1, the order of the Chinese verb phrase "非常 喜欢 C++" is adverb phrase ("非常") + verb phrase ("喜欢 C++"), while the order of the English translation is verb phrase ("like C++") + adverb phrase ("very much"). Therefore, the Chinese production rule $\underline{VP} \rightarrow \underline{ADVP}\ \underline{VP}$ and the corresponding English production rule $VP \rightarrow VP\ ADVP$ cannot be combined directly.



(a) The Chinese parse tree (b) The English parse tree

Figure 3-1: an illustration of constituent reordering between Chinese and English

In vector form, we need to combine the following production rules:

$$\begin{bmatrix} VP \rightarrow ADVP \quad VP \\ VP \rightarrow \quad VP \quad ADVP \end{bmatrix}$$

To solve this problem, symbol ordering must be included into the synchronous grammar. GMTG (Melamed and Wang, 2005) introduces the concept of *precedence array vector* (PAV) to describe such ordering. For example, the above CFG production rules can be combined into one rule in GMTG as:

$$\begin{matrix} VP \\ VP \end{matrix} \rightarrow \bowtie \begin{bmatrix} [1,2] \\ [2,1] \end{bmatrix} \begin{bmatrix} ADVP & VP \\ ADVP & VP \end{bmatrix}$$

On the right hand side of this production rule, symbol $\bowtie$ joins a PAV (the first vector) with a symbol vector (the second vector). In this action, each row in the PAV (called a precedence array) defines the order of the corresponding symbols in the symbol vector. For example, the precedence array and the non-terminal symbols in the second row (i.e. [2,1] and [*ADVP VP*]) are joined to give [*VP ADVP*]. In this way, reordering can be expressed in the production rules.

In a GMTG production rule, each column in the symbol vector represents corresponding symbols in the two languages, and is called a *link*. The left hand side of a GMTG production rule is also a link. In the above production rule, the links are [*VP VP*], [*ADVP ADVP*], and [*VP VP*]. Though each link has the same symbol in this example, a link does not necessarily contain the same symbols. For example, the following production rule was generated during the experiments:

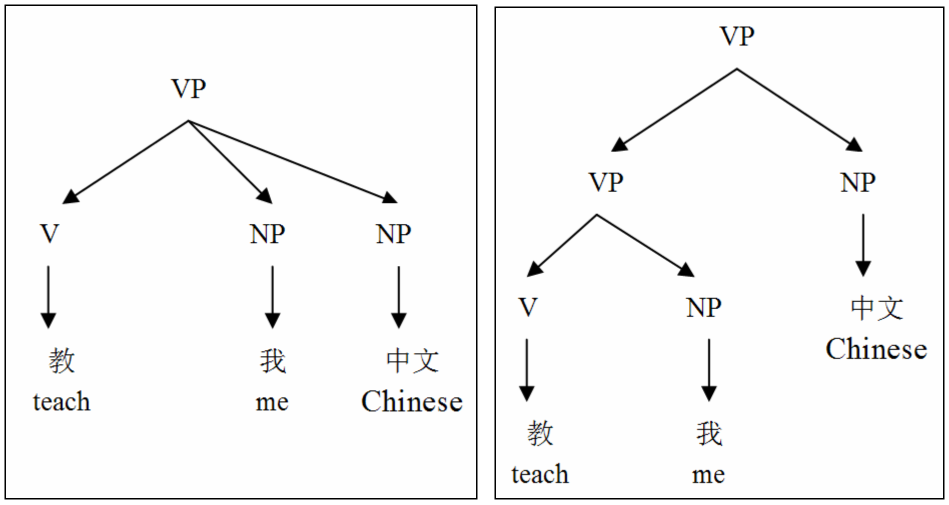
$$\begin{matrix} FRAG \\ NP \end{matrix} \rightarrow \bowtie \begin{bmatrix} [1] \\ [2] \end{bmatrix} \begin{bmatrix} VV & \varnothing \\ \varnothing & NN \end{bmatrix}$$

In the above production, all three links (i.e. [*FRAG NP*], [*VV* ∅] and [∅ *NN*]) have different symbols. Notice that the empty symbol (∅) in a link means that the corresponding symbol in the link does not align to anything.

3.1.1.2 Discontinuity

Section 2.3.1 introduced the CNF for CFG. Because there are at most two symbols on the right hand side of a production rule, CNF can be seen as a *binarised grammar*. By the restriction on production rules, binarised grammars bring simplicity and thus efficiency to parsing algorithms. The comparison of the CYK and the Earley algorithms in Section 2.3.1 is an example. Binarised grammars are widely used in parsing algorithms. The CYK algorithm, the Viterbi algorithm (Section 2.3.2) and the Inside algorithm (Section 2.3.2) all use CNF (see the method *Compose* in each of the above grammars).

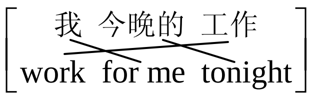
Grammars can be binarised recursively, each rule being processed by inserting intermediate symbols in the right hand side. Figure 3-2 illustrates an example.



(a) Before binarisation (b) After binarisation

Figure 3-2: grammar binarisation

The binarisation of grammar can also break the continuity of phrases, which complicates the situation. For example, the binarisation of the following phrases results in a discontinuous phrase "我…工作", which aligns to "work for me".



In GMTG, such discontinuity can also be expressed by PAV. For example, Figure 3-3 illustrates the GMTG parse tree of the above phrase pair, showing the corresponding PAVs.

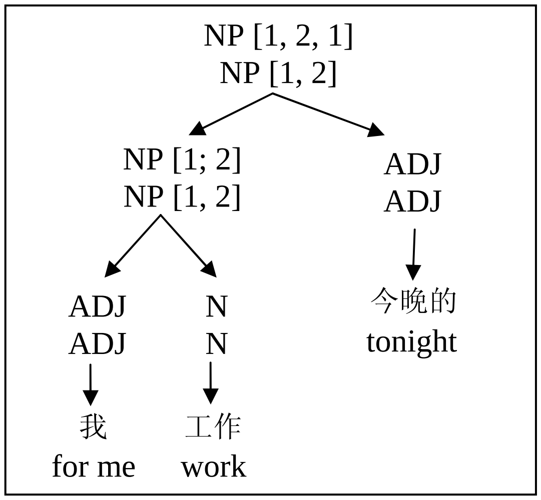


Figure 3-3: alignment with discontinuous symbols

In the above figure, the phrase "我今晚的工作" is broken into the discontinuous phrase "我 ... 工作" and the phrase "今晚的". The order of the two sub phrases is expressed by the corresponding PAV ([1,2,1]) in the production rule:

$$\begin{matrix} NP \\ NP \end{matrix} \rightarrow \bowtie \begin{bmatrix} [1,2,1] \\ [1,2] \end{bmatrix} \begin{bmatrix} NP & ADJ \\ NP & ADJ \end{bmatrix}$$

What is more, the gap in phrase "我 ... 工作" is expressed by the mark ";" in the PAV [1;2] of the following production rule:

$$\begin{matrix} NP \\ NP \end{matrix} \rightarrow \bowtie \begin{bmatrix} [1;2] \\ [1,2] \end{bmatrix} \begin{bmatrix} ADJ & N \\ ADJ & N \end{bmatrix}$$

The total number of discontinuous phrases in a precedence array is called the *fanout* of the precedence array. For example, the fanout of [1,2] is 1, while the fanout of [1;2] is 2.

3.1.1.3 Generalised Chomsky Normal Form

For formal definition, this thesis uses the *Generalised Chomsky Normal Form* (GCNF) (Melamed et al., 2004) to express binarised GMTG, consistent with the software system for the experiments.

Like CNF, GCNF defines two types of production rules – the non-terminating productions, which involve only non-terminal symbols, and the terminating productions, which involve terminal symbols.

The non-terminating productions join a PAV and a symbol vector on the right hand side. Example non-terminating rules can be:

$$\begin{matrix} FRAG \\ NP \end{matrix} \rightarrow \bowtie \begin{bmatrix} [1] \\ [2] \end{bmatrix} \begin{bmatrix} VV & \varnothing \\ \varnothing & NN \end{bmatrix} \quad \text{or} \quad \begin{matrix} NP \\ NP \end{matrix} \rightarrow \bowtie \begin{bmatrix} [1,2,1] \\ [1,2] \end{bmatrix} \begin{bmatrix} NP & ADJ \\ NP & ADJ \end{bmatrix}$$

In comparison, a terminating production has only one terminal symbol link on the right hand side, and therefore no PAV. What is more, it has only one language active, and uses empty symbols (∅) for the other language. Examples include:

$$\begin{matrix} N \\ \varnothing \end{matrix} \rightarrow \begin{matrix} \text{我} \\ \varnothing \end{matrix} \quad \text{or} \quad \begin{matrix} \varnothing \\ \text{V} \end{matrix} \rightarrow \begin{matrix} \varnothing \\ \text{like} \end{matrix}$$

According to the above definition, the GCNF syntax tree for Figure 1-6 is:

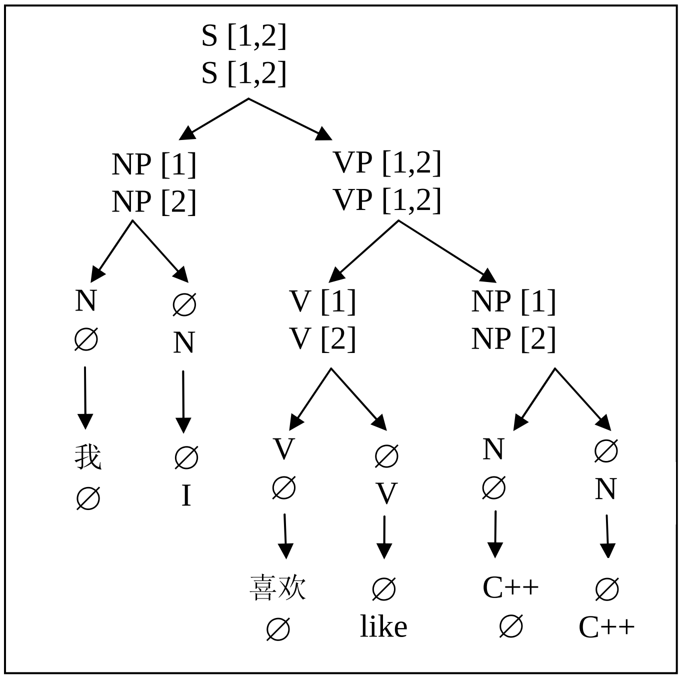


Figure 3-4: the GCNF format of Figure 1-6

In the above figure, there are four non-terminating production rules, which are:

$$\begin{matrix}S\\ S\end{matrix} \to \bowtie \begin{bmatrix}{[1,2]}\\ {[1,2]}\end{bmatrix}\begin{bmatrix}NP & VP\\ NP & VP\end{bmatrix},\quad \begin{matrix}VP\\ VP\end{matrix} \to \bowtie \begin{bmatrix}{[1,2]}\\ {[1,2]}\end{bmatrix}\begin{bmatrix}V & NP\\ V & NP\end{bmatrix},$$

$$\begin{matrix}VP\\ VP\end{matrix} \to \bowtie \begin{bmatrix}{[1]}\\ {[2]}\end{bmatrix}\begin{bmatrix}V & \varnothing\\ \varnothing & V\end{bmatrix} \text{ and } \begin{matrix}NP\\ NP\end{matrix} \to \bowtie \begin{bmatrix}{[1]}\\ {[2]}\end{bmatrix}\begin{bmatrix}N & \varnothing\\ \varnothing & N\end{bmatrix}$$

Meanwhile, there are six terminating production rules, including:

$$\begin{matrix}N\\ \varnothing\end{matrix}\to\begin{matrix}\text{我}\\ \varnothing\end{matrix},\ \begin{matrix}\varnothing\\ N\end{matrix}\to\begin{matrix}\varnothing\\ \text{I}\end{matrix},\ \begin{matrix}V\\ \varnothing\end{matrix}\to\begin{matrix}\text{喜欢}\\ \varnothing\end{matrix},\ \begin{matrix}\varnothing\\ V\end{matrix}\to\begin{matrix}\varnothing\\ \text{like}\end{matrix},\ \begin{matrix}N\\ \varnothing\end{matrix}\to\begin{matrix}\text{C++}\\ \varnothing\end{matrix} \text{ and } \begin{matrix}\varnothing\\ N\end{matrix}\to\begin{matrix}\varnothing\\ \text{C++}\end{matrix}$$

The GCNF of MTG is useful to express a wide range of bilingual alignments. It is adequate as a synchronous grammar between Chinese and English.

3.1.2 The statistical model

This section introduces the statistical model for GMTG. This model defines the method in which parse trees can be scored and ranked.

3.1.2.1 Lexicalisation

Section 2.3.2 shows that lexicalisation is very important for the statistical parsing of CFG. This is because head words bring useful context information to resolve ambiguity and improve accuracy. This is the same for probabilistic GMTG.

GMTG can be lexicalised separately by each language. For each language in the combined grammar, the head word of a non-terminal symbol is the corresponding CFG lexical head. Take the following production rule for example:

$$\begin{matrix}VP[\text{准备}]\\ VP[\text{prepare}]\end{matrix} \to \bowtie \begin{bmatrix}{[1,2]}\\ {[2,1]}\end{bmatrix}\begin{bmatrix}ADVP[\text{提前}] & V[\text{准备}]\\ ADVP[\text{in advance}] & V[\text{prepare}]\end{bmatrix}$$

The head words in each language are decided from the following CFG rules:

$$VP[\text{准备}] \to ADVP[\text{提前}]\ V[\text{准备}] \qquad \text{(Chinese)}$$
$$VP[\text{prepare}] \to V[\text{prepare}]\ ADVP[\text{in advance}] \qquad \text{(English)}$$

In a mono-lingual production rule, the symbol on the right hand side whose head word is propagated to the left hand side is called the *heir*. In the above examples, *V* is the heir of both the Chinese and the English production rule. The index of heir is called the *heir role*. For example, the heir role for the above Chinese and English production rules are 2 and 1, respectively. For a GMTG production rule, combining the heir role for each language makes an *heir role vector*. For example, the heir role vector of the above example is [2,1]. Similarly, combining the symbol heir in each language makes a *heir vector*. In the above example, the heir vector is the second column in the symbol vector: [*V*[准备], *V*[prepare]].

3.1.2.2 Six events to calculate production probability

Similar to PCFG, the probability of a GMTG parse tree can be computed by the overall probability of the production rules that generate it. Suppose that a parse tree is generated by a set of productions $LHS_i \rightarrow RHS_i$. In equation form, the probability of the tree is:

$$P(T \mid S) = \prod_i P(RHS_i \mid LHS_i)$$

Now we define a statistical model by breaking the probability of each production rule into smaller factors, so that it becomes more computable.

Consider the production $LHS \rightarrow RHS$, where *RHS* has *v* links (*v*=1 or *v*=2). When *v*=1, this production is a terminating production (Section 3.1.1.1). In this case, *RHS* is just the lexical heads from *LHS*. Thus $P(RHS \mid LHS,\ v=1) = 1$, and

$$\begin{aligned} & P(RHS,\ v=1 \mid LHS) \\ = \; & P(v=1 \mid LHS)\ P(RHS \mid LHS,\ v=1) \\ = \; & P(v=1 \mid LHS) \end{aligned}$$

When *v*=2, this production is a non-terminating production, and *RHS* consists of a precedence array vector *PAV* and a symbol vector *SYM* (Section 3.1.1.1). Denote the heir role vector with *hrv*, and the corresponding heir vector with *HV* (Section 3.1.2.1). Also, denote the dependencies with *D*. (These dependencies are decided in each dimension separately, in the same way as Section 2.3.2) According to the chain rule, the probability

$$\begin{aligned} & P(RHS,\ v=2 \mid LHS) \\ = \; & P(v=2 \mid LHS)\ P(SYM, PAV \mid LHS, v=2) \\ = \; & P(v=2 \mid LHS)\ P(SYM \mid LHS, v=2)\ P(PAV \mid SYM, LHS, v=2) \\ = \; & P(v=2 \mid LHS)\ P(hrv \mid LHS, v=2)\ P(HV \mid hrv, LHS, v=2) \\ & P(D \mid HV, hrv, LHS, v=2)\ P(PAV \mid SYM, LHS, v=2) \end{aligned} \tag{3-1}$$

Equation 3-1 can be further simplified with independence assumptions. For example, it can be assumed that:

- $P(HV \mid hrv, LHS, v=2) = P(HV \mid LHS)$ (*HV* is only dependent on *LHS*)
- $P(D \mid HV, hrv, LHS, v=2) = P(D \mid LHS)$ (dependencies only depend on *LHS*)
- $P(PAV \mid SYM, LHS, v=2) = P(PAV \mid SYM, LHS)$

In the above equations, *v*=2 is omitted. This is because in the non-terminating rules of the GCNF of GMTG, *v*=2 is a certain condition (Section 3.1.1.3).

While $P(PAV \mid SYM, LHS)$ can be computed directly, it can also be broken into the production of the probabilities of precedence arrays (*PA*) in each dimension on the assumption that they are mutually independent:

$$P(PAV \mid SYM, LHS) = \prod_{dim} P(PA_{dim} \mid SYM_{dim}, LHS_{dim}) \tag{3-2}$$

Both methods to compute $P(PAV \mid SYM, LHS)$ are provided by GenPar. In the experiments of this thesis, Equation 3-2 is used.

3.1.3 Statistical parsing with GMTG

Like CFG, GMTG can be parsed with the generalised parsing algorithm in Section 2.3.3. This algorithm has five components: a grammar, a logic, a semiring, a search strategy and a termination condition. For the particular problem of probabilistic GMTG parsing, the grammar (probabilistic GMTG) and the semiring (non-negative float) are implicitly defined. The following sections define the logic, the search strategy and the terminating condition.

The logic generates new search state items (i.e. partial parse trees) from existing ones. The search strategy determines the order in which state items processed by the logic. Triggered by an initial state item from the search strategy, the logic starts generating new state items, and then passing them to the search strategy. This process loops until the termination condition is met. The parsing goal is normally a (highly scored) parse tree that covers the whole input span.

3.1.3.1 The logic

Similar to the mono-lingual case, bilingual parsing can take the bottom-up chart approach in Section 2.3 (e.g. the CYK algorithm, the Viterbi algorithm and the Inside algorithm). Starting from the input sentence (i.e. terminal symbols), new partial parse trees are generated upwards. This is done by two types of actions: *scan*, which builds up from a terminal symbol by a terminating production rule; and *compose*, which builds up from two non-terminal symbols by a non-terminating production rule.

In the logic, partial parse trees are represented by search state items. Each state item contains the spans in the input sentences that the partial parse tree generates. One span is used for each language in the input sentence pair. Due to grammar binarisation (Section 3.1.1.2), this span can be discontinuous. Discontinuous input spans can be expressed by a list of pairs, indicating a series of continuous sub spans. For example, [(1,2),(3,6)] expresses a discontinuous span that contains two continuous sub spans (one starts from index 1 and has length 1, and the other starts from index 3 and has length 3). If the length of an input sentence is n, the discontinuous span [(1,n+1)] covers the whole sentence.

The difference between PAV and discontinuous span is worth noticing. PAV is a part of GMTG to describe non-terminal symbols, including their discontinuity. In contrast, discontinuous span is used for terminal symbols. It is used only in statistical parsing, recording the index of certain terminal symbols in a sentence.

The compose action generates a new state item by joining two partial parse trees to build a larger one. Correspondingly, the span for the new state item is the total span of the two original items. In the discontinuous case, it is the concatenation of all continuous sub spans in the two original spans. Melamed (2003) uses operator + to express such concatenation. For example:

$$[(1,2),(3,6)] + [(2,3)] = [(1,6)]$$

Moreover, in the compose action, the order of terminal symbols expressed by the discontinuous spans needs to be consistent with the order of corresponding non-terminal symbols expressed by the PAV. The relationship between discontinuous spans and PAVs can be expressed by the relativization operator ⊗ (Melamed, 2003). This operator computes the relative positions of two discontinuous spans in the form of a precedence array. For example:

$$[(1,2),(3,6)] \otimes [(2,3)] = [(1,2,1)]$$

$$[(1,2),(4,6)] \otimes [(2,3)] = [(1,2;1)]$$

Suppose that an input sentence pair is $f_1,\ldots,f_J$ and $e_1,\ldots,e_I$. A bottom-up partial parse tree can consist of only one node containing a terminal symbol link. The corresponding state items can be represented by:

$$\begin{matrix} f_j \\ \varnothing \end{matrix} \text{ or } \begin{matrix} \varnothing \\ e_i \end{matrix}$$

Meanwhile, the state items for higher partial parse trees contain a non-terminal symbol link for the top of the tree, and the discontinuous spans for the input terminal symbols covered by the parse tree. Denote the top of the tree as $[X_1,X_2]$ and the discontinuous spans as $[sx_1,sx_2]$, the state item can be represented by:

$$\begin{bmatrix} X_1 & sx_1 \\ X_2 & sx_2 \end{bmatrix}$$

Given the above representations, the logic for the probabilistic GMTG parsing can be summarised by Table 3-1.

Action	Trigger item	Generated item	Production rule
Scan	$\begin{matrix} f_j \\ \varnothing \end{matrix} \Big/ \begin{matrix} \varnothing \\ e_i \end{matrix}$	$\begin{bmatrix} X & [(j-1,j)] \\ \varnothing & [] \end{bmatrix} \Big/ \begin{bmatrix} \varnothing & [] \\ X & [(i-1,i)] \end{bmatrix}$	$\begin{matrix} X \\ \varnothing \end{matrix} \to \begin{matrix} t \\ \varnothing \end{matrix} \Big/ \begin{matrix} \varnothing \\ X \end{matrix} \to \begin{matrix} \varnothing \\ t \end{matrix}$
Complete	$\begin{bmatrix} Y_1 & sy_1 \\ Y_2 & sy_2 \end{bmatrix}$ and $\begin{bmatrix} Z_1 & sz_1 \\ Z_2 & sz_2 \end{bmatrix}$	$\begin{bmatrix} X_1 & sy_1+sz_1 \\ X_2 & sy_2+sz_2 \end{bmatrix}$	$\begin{matrix} X_1 \\ X_2 \end{matrix} \to \bowtie \begin{bmatrix} \mathrm{sy}_1 \otimes \mathrm{sy}_1 \\ \mathrm{sy}_2 \otimes \mathrm{sz}_2 \end{bmatrix} \begin{bmatrix} Y_1 \; Z_1 \\ Y_2 \; Z_2 \end{bmatrix}$

Table 3-1: the logic for probabilistic parsing with GMTG

3.1.3.2 The search strategy

The search strategy maintains a list of search state items, ranking them by a certain standard. The state item with the highest rank will be passed to the logic for new item generation. The search strategy can be implemented as an agenda, which can be seen as a queue of state items ordered by a certain criterion. For example, an agenda may order items by their current score. With this agenda the search strategy becomes a best first search. Alternatively, the agenda may order items by their size, which is the size of the corresponding partial parse tree. The smaller size an item has, the higher it is ranked. With this agenda the search strategy becomes a bottom-up search. This experiment uses a best-first agenda.

Besides best-first and bottom-up search, A-star search (Russell and Norvig, 2003) can also be used as the search strategy. The difference between A-star search and best-first search is that A-star search ranks an item not by its current score, but by an estimation of its best total score. The best total score of an item is that of the best possible complete parse tree built from the current partial parse tree, and is the combination of the current score and the estimated best score for the rest of the possible total parse tree. The advantage of A-star search is that by accurate estimation of total scores, it is possible to expand a much smaller search space before a goal is found.

3.1.3.3 The termination condition

Apart from the common parsing goal that the whole input is included into a parse tree, there are other termination conditions. For example, it may be necessary to terminate the parsing process if it is taking too much time, or using up the memory resource. It is also possible to have composite termination conditions by using logical conjunctions and disjunctions. For example, a parsing goal may be a parse tree that "starts from a special non-terminal symbol AND covers the whole input span". Or the goal may be "running for 24 hours OR using more than 4GB memory".

3.1.4 The training process

The last two sections introduced the model and the decoding process for the statistical parsing of GMTG. The other important factor is the learning process, where parameters in the statistical model (i.e. probabilities for production rules) are determined. In the mono-lingual case (Section 2.3.2), two different learning methods are used to train PCFG, making use of different forms of corpora. Similarly, the following training methods can be used for bilingual GMTG.

3.1.4.1 Maximum likelihood training

Maximum likelihood estimation can be used when a manually parsed treebank is available (Section 2.3.2). For the training of statistical GMTG, such a treebank enabling maximum likelihood learning must contain manually parsed sentence pairs in GMTG.

Currently there is no existing bilingual GMTG treebank that contains enough parsed sentence pairs to be used in this experiment. However, parallel sentences in Chinese and English are available in reasonable size. What is more, the monolingual structure for both Chinese and English can be derived from existing statistical parsers. Therefore, by combining corresponding monolingual parse trees, it is possible to introduce a bilingual treebank. Such a process is called hierarchical alignment (Melamed and Wang, 2005).

Hierarchical alignment is currently a necessary step to train probabilistic GMTG. After this step, maximum likelihood learning can be performed from the output bilingual treebank. Section 3.2 will show that hierarchical alignment can be done by the generalised parsing process.

3.1.4.2 Expectation maximisation training

As was described in Section 2.3.2, the EM machine learning algorithm can be used to train a probabilistic grammar without a treebank. For statistical GMTG, the training data can be a set of parallel sentence pairs which can be generated by GMTG. Starting with an initial probabilistic grammar, the algorithm updates it by EM iterations over the training sentence pairs. This method converges to a grammar that brings the probability of the training data to a local maximum value.

However, the effect of EM training is dependent on the initial values. When the initial grammar is not well chosen, the local maximum value can be far from the global maximum value. Because of this, the EM training method is too unreliable to be used alone for GMTG training. At the same time, it can be used to further improve an initial grammar that is reasonably near the global optimal value. For example, it can be used for an optimisation after maximum likelihood learning.

3.2 Hierarchical alignment by parsing

The goal of hierarchical alignment is building a bilingual treebank from two mono-lingual treebanks. For each pair of mono-lingual tree, the corresponding symbols are aligned hierarchically to make a combined bilingual tree. In order to achieve this goal, the algorithmic solution is searching for the best aligned bingual tree from all possible alignments.

Without introducing any restrictions, two mono-lingual trees can be combined arbitrarily – one symbol in a mono-lingual tree can be aligned to any symbol in the other. Each possible combination makes a candidate for the bilingual parse tree. Correspondingly, the set of all such combinations makes a search space. In order to find the correctly combined bilingual tree from this search space, we can score each candidate combination, enabling the search goal to get the best score.

This search problem looks like statistical parsing in that it is to find the bilingual tree that has the highest score among many bilingual trees. The difference is that the arbitrarily aligned candidate trees can be not in GMTG at all. In spite of this, we can still borrow experience from the statistical model which calculates the score of a parse tree. As was described in Section 2.3.2 and Section 3.1.2, the score of a parse tree can be seen as the overall score of the production rules that generate it. In this way the calculation of parse tree scores is reduced to the calculation of production scores, which has simpler forms.

Candidates for hierarchical alignment can be scored in the same way, treating each combined tree as if it were generated by "production rules". The score of these pseudo production rules can be calculated by the component mono-lingual grammars. The next two sections describe two methods to score these pseudo production rules, with different information from the mono-lingual grammars.

3.2.1 Calculating combined pseudo production rules from one mono-lingual grammar

This section describes the calculation of combined pseudo production rules by only one mono-lingual grammar, which is comparatively simpler. Suppose that the first language is used. In this case, all non-terminal symbols in the second language can be represented by a dummy non-terminal symbol *NULL*.

In the GCNF format (Section 3.1.1.3), there are two kinds of production rules – the terminating production rules and the non-terminating production rules. A terminating production rule contains only one link on the right hand side. In the lexicalised form, it can be written as:

$$LHS \rightarrow RHS \;=\; \begin{matrix} X_1[h_1] \\ \varnothing \end{matrix} \rightarrow \begin{matrix} h_1 \\ \varnothing \end{matrix} \quad \text{or} \quad \begin{matrix} \varnothing \\ X_2[h_2] \end{matrix} \rightarrow \begin{matrix} \varnothing \\ h_2 \end{matrix} \qquad (3\text{-}3)$$

In Equation 3-3, $X_d[h_d]$ represents a non-terminal symbol X_d whose head word is h_d. For the terminating production rules, only one language is active (As indicated by Equation 3-3, the inactive language has empty symbol $\varnothing$ on both sides of the production rule; see also Section 3.1.1.3). Correspondingly, the scores of combined pseudo production rules are decided by the active language. Therefore, when the first language is active, $P(RHS|LHS)=P(h_1|X_1[h_1])$ – the probability is the same as the corresponding mono-lingual production score. Similarly, when the second language is active, $P(RHS|LHS)=P(h_2|X_2[h_2])$. However, because only the first language is used in this method, we have $P(h_2|X_2[h_2])=1$ only when $X_2=NULL$, and $P(h_2|X_2[h_2])=0$ otherwise.

A non-terminating production rule consists of a PAV and a symbol vector on the right hand side. A lexicalised non-terminating production rule can be written as:

$$LHS \rightarrow RHS \;=\; \begin{matrix} X_1[h_1] \\ X_2[h_2] \end{matrix} \rightarrow \;\bowtie \begin{bmatrix} pa_1 \\ pa_2 \end{bmatrix} \begin{bmatrix} Y_1[g_1]\; Z_1[h_1] \\ Y_2[g_2]\, Z_2[h_2] \end{bmatrix} \tag{3-4}$$

In the above equation, X_d, Y_d and Z_d are non-terminal symbols, h_d and g_d are head words and pa_d are precedence arrays. Therefore, $P(RHS|LHS)$ can be written as $P(pa_1, pa_2, Y_1, Y_2, Z_1, Z_2, g_1, g_2 \mid X_1, X_2, h_1, h_2)$. Using the chain rule, this probability can be broken down as follows:

$$\begin{aligned} & P(pa_1, pa_2, Y_1, Y_2, Z_1, Z_2, g_1, g_2 \mid X_1, X_2, h_1, h_2) \\ = \;& P(pa_1, g_1, Y_1, Z_1 \mid X_1, X_2, h_1, h_2) \times \\ & P(Y_2, Z_2 \mid pa_1, g_1, Y_1, Z_1, X_1, X_2, h_1, h_2) \times \\ & P(g_2 \mid pa_1, g_1, Y_1, Y_2, Z_1, Z_2, X_1, X_2, h_1, h_2) \times \\ & P(pa_2 \mid pa_1, g_1, g_2, Y_1, Y_2, Z_1, Z_2, X_1, X_2, h_1, h_2) \end{aligned} \tag{3-5}$$

To further simplify the calculation of Equation 3-5, the following independence assumptions can be made:

(1) $P(pa_1, g_1, Y_1, Z_1 \mid X_1, X_2, h_1, h_2) = P(pa_1, g_1, Y_1, Z_1 \mid X_1, h_1)$ (mono-lingual production)

(2) $P(Y_2, Z_2 \mid pa_1, g_1, Y_1, Z_1, X_1, X_2, h_1, h_2) = \begin{cases} 1 \text{ if } Y_2 = Z_2 = NULL \\ 0 \text{ otherwise} \end{cases}$ (the unused language)

(3) $P(g_2 \mid pa_1, g_1, Y_1, Y_2, Z_1, Z_2, X_1, X_2, h_1, h_2) = P(g_2 \mid g_1)$ (word-to-word probability)

(4) $P(pa_2 \mid pa_1, g_1, g_2, Y_1, Y_2, Z_1, Z_2, X_1, X_2, h_1, h_2) = P(pa_2) = \dfrac{1}{\mu(f)}$, where $\mu(f)$ is the number of unique precedence arrays of the maximum fanout f.

Thus the simplified calculation for the pseudo non-terminating production rules is as follows:

$$P(RHS \mid LHS) = \begin{cases} \dfrac{P(pa_1, g_1, Y_1, Z_1 \mid X_1, h_1) P(g_2 \mid g_1)}{\mu(f)} & \text{, if } Y_2 \text{ and } Z_2 \text{ are both } NULL \\ 0 & \text{, otherwise} \end{cases} \quad (3\text{-}6)$$

3.2.2 Calculating combined pseudo production scores from two mono-lingual grammars

Moving from the simpler case, this section describes the calculation of combined pseudo production rules from both mono-lingual grammars.

Similar to Section 3.2.1, the scores for combined pseudo terminating production rules are calculated by the active language. When the first language is active, $P(RHS|LHS)=P(h_1|X_1[h_1])$, and this probability is the same as the corresponding mono-lingual production rule. Similarly, when the second language is active, $P(RHS|LHS)=P(h_2|X_2[h_2])$, and this probability is the corresponding mono-lingual production score of the second language.

For combined non-terminating production rules, the calculation can be based on Equation 3-6. In a simple form, the production score of the second language can be incorporated into the combined production score as a factor. In equation form:

$$\begin{aligned} & P(RHS \mid LHS) \\ = \; & P(pa_1, pa_2, Y_1, Y_2, Z_1, Z_2, g_1, g_2 \mid X_1, X_2, h_1, h_2) \\ = \; & \frac{P(pa_1, g_1, Y_1, Z_1 \mid X_1, h_1) P(pa_2, g_2, Y_2, Z_2 \mid X_2, h_2) P(g_2 \mid g_1)}{\mu(f)} \end{aligned}$$

3.2.3 Hierarchical alignment by generalised parsing

Section 3.2.1 and Section 3.2.2 define possible ways to calculate the combined production scores. From these production scores, the score of combined bilingual trees can be calculated, and the best tree can be chosen by the search algorithm.

In summary, the input for the hierarchical alignment problem is a set of aligned bilingual sentences, the output is a bilingual parse tree, and the search process is based on the calculation of candidate trees by production scores. All the above conditions are the same as the bilingual parsing problem, enabling hierarchical alignment to be implemented by generalised parsing.

Among the five generalised parsing components, the semiring of hierarchical alignment is the same as that of bilingual parsing (Section 3.1.3) – both are non-negative floating point numbers as probabilities. However, the grammar for hierarchical alignment is different from the grammar for bilingual parsing (i.e. probabilistic GMTG). In fact, the hierarchical alignment process is not based on any bilingual grammar – as stated in the beginning of Section 3.2, the candidate trees are arbitrarily combined. However, because scores are assigned to the pseudo bilingual production rules, it can be regarded as using a pseudo grammar. This pseudo grammar does not contain any production rules, but with an input symbol, it can generate possible production rules and assign scores to them, as described in Section 3.2.1 and Section 3.2.2.

The external behaviour of the pseudo grammar can be made exactly the same as probabilistic GMTG, so that hierarchical alignment can use the same logic and search strategy as bilingual parsing (Section 3.1.3.1 and Section 3.1.3.2). What is more, because the parsing goals of the two problems are the same, hierarchical alignment can also use the terminating conditions in Section 3.1.3.3.

3.3 Translation by bilingual parsing

This section describes the algorithm of SMT by GMTG parsing, using the generalised parsing algorithm. As introduced in Section 1.1.4, the process of SMT by parsing is a parsing process, which derives a bilingual parse tree. Given a probabilistic GMTG model, the main differences between the SMT by parsing algorithm and the GMTG parsing algorithm in Section 3.1.3 include: (1) the inputs for translation are mono-lingual sentences instead of bilingual sentence pairs; (2) translation sentence pairs need to be extracted from the output parse trees.

It is possible to implement the decoding process of translation by generalised parsing. Comparing the five generalised parsing components, translation by parsing has the same grammar (probabilistic GMTG) and semiring (non-negative floating point numbers) as bilingual parsing. Not considering the flattening of parse trees, the termination condition and parsing goals are also the same.

The logic for translation needs to be different from the one for bilingual parsing (Section 3.1.3.1). This is mainly because the input for translation contains only one language. During the translation by parsing process, terminal symbols of the target language are not available from input. In order to build bilingual GMTG parse trees, a special action is needed to bring terminal symbols of the other language into search state items.

One possible choice is the *load* action (Melamed and Wang, 2005), which is similar to the *scan* action in that it generates new items according to terminating productions. However, the *load* action does not take any inputs. Instead of matching input terminal symbols with terminating productions, it allows state items to be generated from any terminal symbols. Using a similar format as Table 3-1, Table 3-2 illustrates the logic for translation.

Action	Trigger item	Generated item	Production rule
Scan	$\begin{matrix} f_j \\ \varnothing \end{matrix}$	$\begin{bmatrix} X ; [(j-1, j)] \\ \varnothing ; \end{bmatrix}$	$\begin{matrix} X \\ \varnothing \end{matrix} \rightarrow \begin{matrix} t \\ \varnothing \end{matrix}$
Load	None	$\begin{bmatrix} \varnothing ; [] \\ X ; \end{bmatrix}$	$\begin{matrix} \varnothing \\ X \end{matrix} \rightarrow \begin{matrix} \varnothing \\ t \end{matrix}$
Complete	$\begin{bmatrix} Y_1 ; sy_1 \\ Y_2 ; \end{bmatrix}$ and $\begin{bmatrix} Z_1 ; sz_1 \\ Z_2 ; \end{bmatrix}$	$\begin{bmatrix} X_1 ; sy_1 + sz_1 \\ X_2 ; \end{bmatrix}$	$\begin{matrix} X_1 \\ X_2 \end{matrix} \rightarrow \bowtie \begin{bmatrix} \mathrm{sy}_1 \otimes \mathrm{sy}_1 \\ pa_2 \end{bmatrix} \begin{bmatrix} Y_1 \; Z_1 \\ Y_2 \; Z_2 \end{bmatrix}$

Table 3-2: the logic for translation by parsing

It can be seen from the table that items in the translation target language do not contain any span information. This is because no input sentences in the language are available. As a consequence, there is no need to choose production rules that match the discontinuous spans in this language. pa_2 in the production rule of the "complete" action denotes any precedence array in that dimension.

The main differences between the translation logic and the bilingual parsing logic are the load action and the span information for the translation target language. Besides these differences, the logics for the two problems are the same. For the translation case, although all possible terminal symbols for one language are loaded, the search process will finally determine the best parse tree and remove the unlikely items by scores.

A disadvantage of the load action is that a considerable number of items are introduced into the search space, which will impact decoding efficiency. A possible improvement for the load action is to generate items only from those terminal symbols that are possible translations of the terminal symbols from the source sentence. This will reduce the search space considerably, because the possible word translations for an input sentence are much fewer than all the words from the vocabulary.

Finally, corresponding to the above logic definition, translation by parsing can also use the same search strategy as bilingual parsing, such as the best-first agenda approach.

Chapter 4

The details of experiments

4.1 Introduction

The above theory is applied to Chinese-English SMT by this thesis with large-scale experiments. This chapter records the details of the experiments, including the software systems, the training and testing corpora, and the typical process that is used by all the experiments of this thesis. The output and research questions are discussed in Chapter 5.

The system for the experiments is built upon existing pieces of software. The engineering work includes the choosing and compiling of the software systems and libraries, the selecting and formatting of corpora, the code analysis in accordance with the theory of the last two chapters, the software development work to combine and coordinate different software systems, and the application of automatic MT evaluation methods. One of the challenges of the experiments is training the system with significantly large amounts of data within a reasonable time frame; the techniques used include filtering dispensable time consuming data, running tasks in parallel, and doing experiments incrementally.

4.2 Software

4.2.1 GenPar – the software framework for generalised parsing

GenPar (Burbank et al., 2005) is the major software framework used by the experiments. It provides an implementation of the generalised parsing algorithm. GenPar is written in C++. It was developed during the 2005 Johns Hopkins Language and Speech Processing Workshop.

4.2.1.1 Design and data structures

Figure 4-1 shows the main classes that GenPar uses.

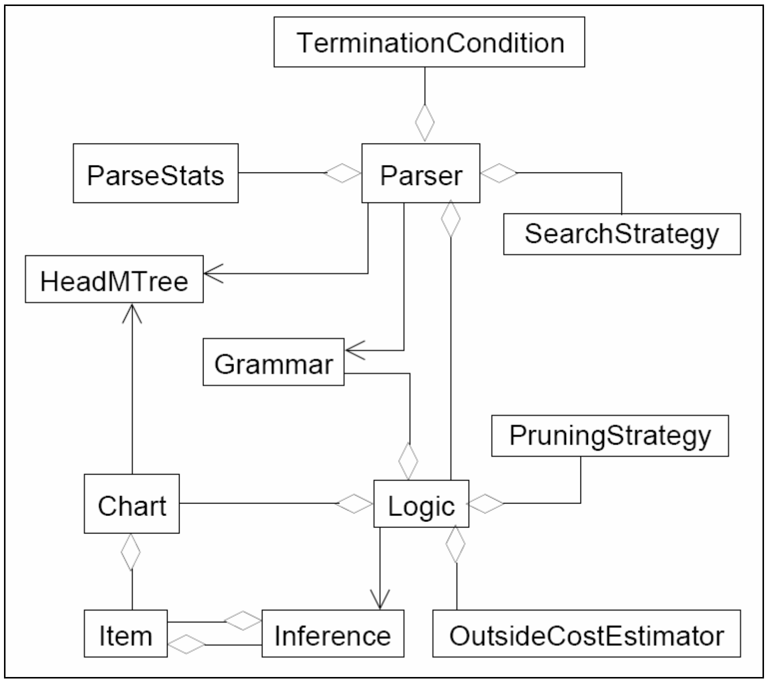


Figure 4-1: design of GenPar; adapted from (Burbank et al., 2005)

According to the five components of the generalised parsing algorithm (Section 2.3.3), GenPar contains five major component classes: `Parser`, `Grammar`, `Logic`, `SearchStrategy` and `TerminationCondition`.

The core component is `Parser`, which contains an implementation of the generalised parsing algorithm in Figure 2-8. By sub-classing the abstract base classes `Grammar`, `Logic`, `SearchStrategy` and `TerminationCondition` and providing detailed implementations, GenPar can be instantiated into specific parsers. For example, using the probabilistic GMTG, the bottom-up logic, the agenda search strategy and the termination condition described in Section 3.1.3 for `Grammar`, `Logic`, `SearchStrategy` and `TerminationCondition`, GenPar can be instantiated as sa bilingual GMTG parser.

GenPar provides a set of implementations of the abstract components. For example, the abstract class Logic is implemented by sub-class `BottomUpLogic` and `BottomUpTranslationLogic`, which can be used for the bottom-up bilingual parsing algorithm (Section 3.1.3.1) and the translation algorithm

(Section 3.3), respectively. The class diagram is shown in Figure 4-2.

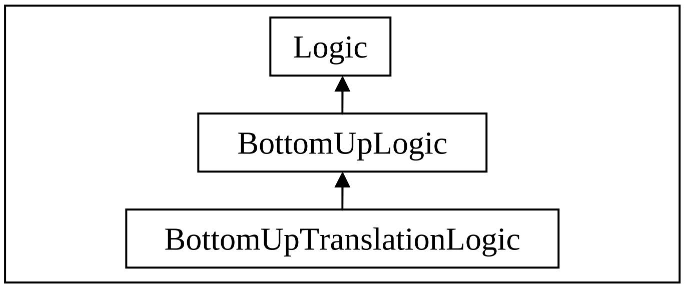


Figure 4-2: classes for the Logic component, adapted from (Burbank et al., 2005)

GenPar supports the use of a set of configuration files to specify which sub-classes are to be used for the abstract components. For example, a possible set of configuration files is shown in Figure 4-3.

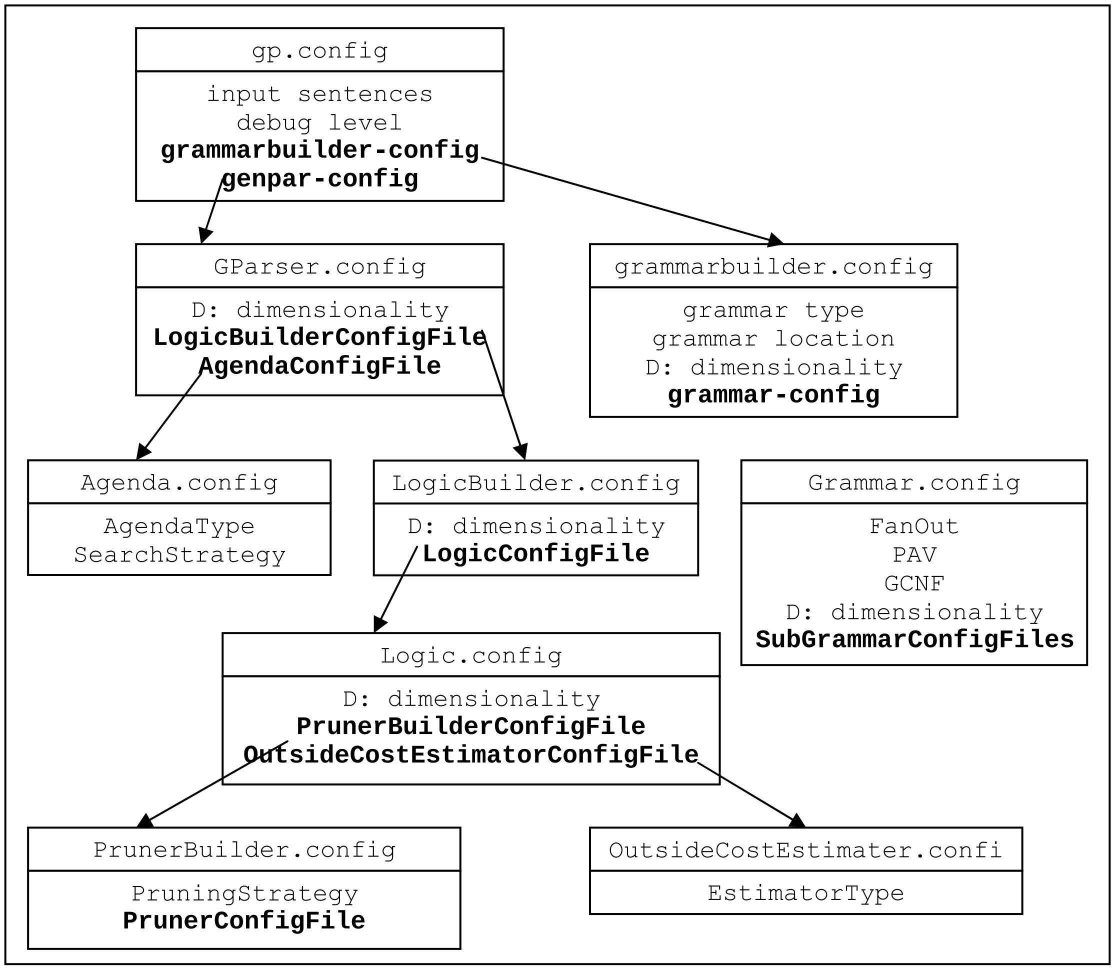


Figure 4-3: GenPar configuration file hierarchy, adapted from (Burbank et al., 2005).

In the above figure, each box represents a file. The first column contains the file name, and the second column contains configuration items in the file. A file can refer to other files, as indicated by links.

4.2.1.2 The typical process for translation by GenPar

As introduced earlier, SMT by parsing is achieved by two steps – training and decoding. To train a statistical GMTG, a corresponding bilingual treebank is needed (Section 3.1.4). This bilingual treebank is not directly available, and can be derived from mono-lingual syntax trees using the hierarchical alignment algorithm (Section 3.2), which requires word-to-word translation probabilitiess. After maximum likelihood training using the bilingual treebank, EM based training (Section 3.1.4.2) can be conducted to optimise the probabilistic grammar. In the above process, most algorithms can be implemented by generalised parsing.

Therefore, a typical process to train a probabilistic GMTG using GenPar includes:

(1) Input: provide the training data.
(2) Pre-process: format the training data and provide mono-lingual treebanks.
(3) Word alignment: provide the word-to-word relationships for the training data.
(4) Hierarchical alignment: derive the bilingual treebank.
(5) Initialise grammar: maximum likelihood training from the bilingual treebank.
(6) Optimise: EM training from the initial grammar (this step is not necessary, but can be used to improve grammar quality).

When training is done, translation by parsing can be done using the generalised parsing algorithm in Section 3.3. Specifically, the following steps can be used:

(1) Input: provide the testing data.
(2) Preprocess: format the testing data.
(3) Translate: do translation by the generalised parsing algorithm in Section 3.3.

After translation, the results are evaluated by the methods from Section 2.2.

4.2.1.3 Existing experiments by GenPar

The GenPar framework was developed in the 2005 Johns Hopkins Workshop. By the time of this thesis, GenPar had been used for experiments of Arabic-English and French-English SMT by parsing. Each language pair requires specific processing, and the overall French-English translation was more accurate than the Arabic-English translation (Burbank et al., 2005). In Section 5.1.1.3, the Chinese-English SMT by parsing results of this thesis are compared with the results of existing Arabic-English and French-English experiments with GenPar.

4.2.2 The Bikel statistical (mono-lingual) parser

The Bikel parser (Bikel, 2002) is a mono-lingual parser that supports different types of statistical parsing models. It is used in the experiments to produce English and Chinese mono-lingual grammars. This parser is implemented in Java.

The Bikel parser requires the part-of-speech information for each word in the input sentence. Therefore, before using the parser, *POS tagging* (Jurafsky and Martin, 2000) is required, which assigns POS tags to the input words.

4.2.3 The LingPipe libraries for the Chinese segmentation task

The LingPipe natural language processing library is used by this thesis for Chinese word segmentation (Section 1.1.5). It is chosen because of its comparatively high accuracy. LingPipe uses an n-gram based language model. It does Chinese word segmentation by a spelling correction algorithm. The library is written in Java.

4.2.4 The Stanford statistical tagger

The Stanford Tagger (Toutanova and Manning, 2000) is used by this thesis for the POS tagging of both Chinese and English. It is based on the maximum entropy probabilistic model (Ratnarparkhi, 1997). An advantage of this model is that it allows different features to be selected in the calculation of tagging probability. The Stanford Tagger is written in Java.

4.2.5 GIZA++ – the word alignment tool

GIZA++ (Och and Ney, 2000) is a general word alignment tool. It is used by this thesis to obtain word-to-word translation probabilities between Chinese and English. It is based on the word alignment models in Section 2.1.2, and it incorporates many features. GIZA++ is written in C++.

4.3 Training and testing data

4.3.1 Training data

Hong Kong Parallel Text corpus is used for the bilingual training. It is produced by the Linguistic Data Consortium (LDC). Its catalog number is LDC2004T08 and it has ISBN 1-58563-290-2. The corpus contains parallel articles in Chinese and English, together with specifications of the (many-to-many) sentence alignment in each pair of articles. There are three collections in the corpus, containing 59 million Chinese words and 49 million English words. Only the News collection is used for this thesis, which contains 27 million Chinese words and 15 million English words (in 2,681 thousand Chinese sentences and 2,952 thousand English sentences, respectively).

The Chinese Treebank version 4.0 is used as the Chinese mono-lingual grammar treebank. It is produced by LDC. Its catalog number is LDC2004T05 and it has ISBN 1-58563-287-2. The corpus contains a collection of Chinese articles, in which each sentence is manually segmented, POS tagged and annotated with a CFG parse tree. There are 664,633 characters (in 15,162 sentences) in the corpus.

The English Treebank version 3.0 is used as the English mono-lingual grammar treebank. It is produced by LDC. Its catalog number is LDC99T42 and it has ISBN 1-58563-163-9. The corpus contains a collection of English articles, from which each sentence is manually POS tagged and parsed. The whole corpus contains over a million words, while this thesis uses only the Wall Street Journal (WSJ) sub collection.

Three corpora provided by the first International Chinese Word Segmentation Bakeoff (from Academia Sinica, Hong Kong City University and Beijing University) are used for the Chinese segmentation task. These corpora contain segmented Chinese sentences. There are in total over 7 million words in the corpora.

4.3.2 Testing data

The Hong Kong Parallel Text corpus is used for part of the testing data. Parallel sentences in the corpus that are not used for training are extracted for testing.

The Multiple-Translation Chinese (MTC) Part 3 corpus is used for another part of the testing data. It was produced by LDC. Its catalog number is LDC2004T07 and it has ISBN 1-58563-289-9. The corpus contains 100 Chinese articles, each having four independent English translations. The corpus is sentence-aligned, and there are 935 Chinese sentences in total.

4.4 Details for a typical experiment

Following the process of Section 4.2.1.2, this section records the detailed process of a typical experiment, including 6 steps for training, 3 steps for translation and 1 step for evaluation. This process is used by most of the experiments.

4.4.1 Input for training

This section describes the details of training step (1) given in Section 4.2.1.2. Two files are produced in this step (`L1.text` and `L2.text`), each containing the training sentences in Chinese and English, respectively. The *n*th sentence in one file is the corresponding translation of the *n*th sentence in the other.

The source files for this step are the Hong Kong Parallel Text data, which are formatted in loose XML, and separated into three folders containing Chinese text, English text and alignment information. A Python script (`corpus_rfmt.py`) is made to extract the corpus data and format them for output. During the process, we made two simplifications: (1) When there are sentences that are not in one-to-one relationship (e.g. one English sentence corresponds to multiple Chinese sentences), they are discarded. (2) When there are quotations (i.e. there are Chinese quotations in the English version), they are discarded from both sides.

Because the Kong Hong Parallel Text is in traditional Chinese (BIG5 encoded), it is first translated into simplified Chinese (GB2312 encoded) so that it can be used consistently with the Chinese Treebank. The translation is table-based[1], because the two character sets are almost many-to-one related. After this, all brackets ("(", ")") are converted to the Chinese version (GB-2312 encoded). This is to avoid clashing with the Stanford tagger, which uses brackets to hold tag information.

[1] See http://freshmeat.net/projects/autoconvert/ for more details.

4.4.2 Pre-processing for training

This section describes the details of training step (2) in Section 4.2.1.2. The input files of this step are the input texts from Section 4.4.1 (`L1.text` and `L2.text`). The productions of this step include two mono-lingual treebanks (`L1.tb` and `L2.tb`), two files containing segmented input Chinese text and tokenised input English text (`L1.tok.text` and `L2.prep.text`), as well as formatted files containing tokenised text (`output.snt` and `output.snt`). This step also produces interim outputs, such as tagged Chinese and English sentences.

4.4.2.1 Mono-lingual treebanks

The Bikel parser is used to produce mono-lingual treebanks for both Chinese and English. It is trained with the Chinese Treebank 4 and the English Treebank 3, respectively. Before training, the files from the Chinese Treebank are switched from "`fid`" format to "`mrg`" format for the parser. This is done by making a Python script (`fid2mrg.py`). After training, the output models are saved into `ctb.obj.gz` (Chinese) and `etb.obj.gz` (English).

With the trained models, the Bikel parser can be used for mono-lingual parsing. Because the Bikel parser requires input sentence with POS information, both the Chinese (`L1.text`) and the English (`L2.text`) sentences need to be POS-tagged. Besides, the input Chinese sentences also need to be segmented.

LingPipe is used for Chinese segmentation. Three corpora provided by the first International Chinese Word Segmentation Bakeoff teams (i.e. Academia Sinica, Hong Kong City University and Beijing University) are converted into simplified Chinese (GB2312 encoded), merged into one file, and then used to train the segmentor. The Chinese sentences in `L1.text` are segmented and saved into `L1.tok.text`. English sentences do not need segmentation. However, they also need tokenisation (e.g. separating punctuation marks from words by space). This task is also performed by the Stanford tagger, and processed together with POS tagging. The tokenised English sentences are saved as `L2.prep.text`.

Both Chinese and English are tagged using the Stanford statistical tagger. The tagger is trained with tagged corpora from the Chinese Treebank 4 and the English Treebank 3, respectively. Before training, corpus data for each language are merged into one file, and transferred into the Stanford tagger file format. This is done by making two Python scripts (`cn_tag_form.py` and `en_tag_form.py`).

After training the tagger, we prepare the segmented Chinese file (`L1.tok.text`) and the input English file (`L2.text`) for tagging. For example, the newest version of the Stanford tagger supports Chinese by Unicode characters. However, the input Chinese text is encoded in GB-2312. Thus the input text (`L1.tok.text`) is first switched to Unicode (by making `gb2utf.py`), and then the tagged text is switched back to GB-2312 (by making `utf2gb.py`). After tagging, the output tagged Chinese and English sentences are saved into `L1.tagged.text` and `L2.tagged.text`, respectively.

The tagged sentences are passed to the Bikel parser, which has just been trained. The output for the parser needs to be configured so that it will include lexical head information, which is necessary for the hierarchical alignment process. The following line is added to the parser's setting file:

```
parser.decoder.outputHeadLexicalizedLabels=true
```

The above configuration is required for both Chinese and English parsing. For English parsing, an alternative choice is using the GenPar add-on setting file `ws05Collins.properties` instead of the default `collins.properties` file.

The parsing process requires significant computational resource. For the aim of producing novel result for Chinese-English translation by parsing, the training data set is expected to be as large as possible. In order to derive parsed data within the time frame for this thesis, the tagged data are divided into 32 groups and processed on 32 separate machines, including jet1 and jet2, as well as booth3 – booth32 in the Thom building. Parsing output files are then merged into one file (`L1.text.tb` for Chinese and `L2.text.tb` for English). Furthermore, earlier parsing outputs are used for translation experiments immediately, before all tagged sentences are parsed. In general, the GenPar translation experiments were performed with smaller training data simultaneously as larger training data was being prepared. It took around 2 months for all the sentences to be parsed.

After parsing, mono-lingual treebanks are extracted from the output parse trees for hierarchical alignment. In each experiment, a specific number of parse trees are extracted from the parser's output (by making `extract_trees.py`). During this process, the parse trees are filtered and unhelpful data are discarded. For example, when the Bikel's parser fails to parse a sentence, it produces "`null`" as the output. These null parse trees in either Chinese or English are removed, together with their corresponding translation. Also, to improve time and space efficiency, sentences that are longer than 30 words are filtered out. Such sentences are comparatively few, and testing shows that the GenPar framework is exponentially inefficient with such sentences during hierarchical alignment.

Before the extracted parse trees are used by GenPar, the terminal symbols are replaced by integers, so that the generalised parsing process is more efficient. This is done by a Perl script (`integerize_trees`) provided by GenPar. The final Chinese and English parse trees are saved to `L1.tb` and `L2.tb`, respectively.

4.4.2.2 Formatted sentence pairs and mono-lingual sentences

For the use in the next steps, segmented Chinese sentences and tokenised English sentences are grouped into one file, where each Chinese sentence is paired with the corresponding English translation by `<s>` and `</s>` tags.

This is achieved by making a Python script (`weavefiles.py`), which reads sentences from multiple files and groups corresponding lines by `<s>` and `</s>` tags. The input files for this script are the segmented Chinese text (`L1.tok.text`) and tokenised English text (`L2.prep.text`) from Section 4.4.2.1, and the output file is called `output.snt`.

`weavefiles.py` is also applied to Chinese mono-lingual sentences, and the output texts are saved at `output.snt.1D`.

4.4.3 Word alignment

This section describes the details of training step (3) in Section 4.2.1.2. The input files for this step are `L1.tok.text` and `L2.prep.text` from Section 4.4.2. This step produces a word-to-word alignment model file (`links`).

This step is based on the theory of Section 2.1.2. GIZA++ is used to derive word-to-word alignments. The detailed steps are shown as follows:

(1) Run `plain2snt.output` (which is compiled together with the GIZA++ package) on `L1.tok.text` and `L2.prep.text`, and get the following output files:

    `L1.tok.text.vcb` (vocabulary file for Chinese)

`L2.prep.text.vcb` (vocabulary file for English)
`L1.tok.text_L2.prep.text.snt` (integerised sentence alignment file)

This step replaces words with integers for both Chinese and English, so that they will be processed more efficiently.

(2) Run `mkcls` (which is a separate package from GIZA++) on `L1.tok.text` and `L2.prep.text` and generate word classes `L1.tok.text.vcb.classes` (for Chinese) and `L2.prep.text.vcb.classes` (for English).

This step is optional. It does bilingual word classification (i.e. group words into equivalent classes – not studied in this thesis; Och, 1999).

(3) Run GIZA++:

```
GIZA++ -S L2.prep.text.vcb -T L1.tok.text.vcb \
       -C L2.prep.text_L1.tok.text.snt –o L1toL2
```

The output file `L1toL2.A3.final` contains the word-to-word probabilities $P(f\,|\,e)$ (Section 2.1.2), where e represents an English word and f represents a Chinese word.

After the above process, the output of GIZA++ is turned into the file format that GenPar can use. This is done by a Perl script provided by GenPar (`giza2ww`).

This thesis also adapts a bi-directional alignment method, which was used by existing Arabic-English and French-English translations by GenPar (Burbank et al., 2005). This process includes the following steps:

(1) Use GIZA++ to generate alignment in both Chinese-English and English-Chinese direction.

(2) Take the intersection of alignments for each sentence in step (1) as the training set for the output alignment.

(3) Using the maximum likelihood estimation, calculate the joint-alignment probability as:

$$P(c_i, e_j) = \frac{Count(c_i, e_j)}{\sum_{c_x, e_y} Count(c_x, e_y)}$$

In the above equation, c_i and e_j are a Chinese word and an English word that are aligned together, while c_x represents any Chinese word in the training set and e_y represents any English word in the training set.

(4) Use all word pairs that have non-zero probability from step (3) as the word

alignment model.

In the above steps, step (2) can take the union or intersection of alignments from step (1). This thesis chooses to use intersection for the efficiency of later steps.

4.4.4 Hierarchical alignment

This section describes the details of training step (4) in Section 4.2.1.2. The input data for this step are the parsed mono-lingual sentences from Section 4.4.2.1 (`L1.tb` and `L2.tb`), the formatted sentences from Section 4.4.2.2 (`output.snt`), as well as the word-to-word alignment models from Section 4.4.3 (`links`). This step produces a hierarchically aligned bilingual treebank file (`tb.out`).

This step is based on the theory of Section 3.2. The hierarchical alignment process is implemented by generalised parsing. A pseudo grammar is used to score candidate combined trees, according to mono-lingual grammars and word-to-word alignments. The pseudo grammar is provided by GenPar, and was specifically designed so that apart from this grammar, all the other components for hierarchical alignment are the same as for bilingual parsing. The most important configuration files for GenPar hierarchical alignment are shown in Table 4-1.

Configuration file	Definition	Value
`grammarbuilder.config`	Grammar	`PseudoMTG` (the pseudo grammar for hierarchical alignment, Section 3.2.1 and Section 3.2.2)
	Sub grammar 0	`L1.tb` (Chinese grammar)
	Sub grammar 1	`L2.tb` (English grammar)
`grammar.config`	Word alignment	`links` (Word-to-word alignments)
`LogicBuilder.config`	Logic	`C` (the bilingual parsing logic, Section 3.2.3)

Table 4-1: the GenPar configuration files for hierarchical alignment

4.4.5 Initialise grammar

This section describes the details of training step (5) in Section 4.2.1.2. The input files for this step are the formatted sentence pairs from Section 4.4.2 (`output.snt`) and the bilingual treebank from Section 4.4.4 (`tb.out`). This step produces a probabilistic grammar model (`pmtg.mle`).

This step is based on the theory of Section 3.1.4.1. Bilingual trees from `tb.out` are scanned, and the production probabilities are estimated by the maximum likelihood principle.

This step is achieved using the GenPar program `trees2grammar`.

4.4.6 Optimise

This section describes the details of training step (6) in Section 4.2.1.2. The input files for this step are the formatted sentence pairs from Section 4.4.2 (`output.snt`) and the initialised grammar from Section 4.4.5 (`pmtg.mle`). This step produces the optimised probabilistic grammar model (`model_final`).

This step is based on the theory from Section 3.1.4.2. The output grammar from the last step is taken as the initial value, and several EM iterations are taken to update the grammar to a local optimal value.

This step is achieved using the GenPar program `treeLearn`.

4.4.7 Input for translation

This section describes the details of translation step (1) in Section 4.2.1.2. Two files are produced in this step (`L1.text` and `L2.text`), each containing testing sentences in Chinese and English, respectively. The *n*th sentence in one file is the corresponding translation of the *n*th sentence in the other.

`L1.text` and `L2.text` make a testing set, where `L1.text` is the source text for translation and `L2.text` is the reference translation. Two testing sets are prepared using separate corpora. Firstly, parallel sentences from Hong Kong Parallel Text that are not used as training data are extracted as a test set. This is done by making a Python script (`make_test_data.py`). 983 sentence pairs are extracted as the testing data. Secondly, in order to test different writing styles, Multiple-Translation Chinese (MTC) Part 3 is used as another test set. There are in all 935 sentences in this corpus, which is comparable to the first test set.

4.4.8 Pre-process for translation

This section describes the details of translation step (2) in Section 4.2.1.2. The input files of this step are the testing text from Section 4.4.7 (`L1.text` and `L2.text`). The productions of this step include two files containing segmented input Chinese text and tokenised input English text (`L1.prep.text` and `L2.prep.text`), as well as two formatted files containing the tokenised text (`output.snt` and `output.snt.1D`).

Similar to Section 4.4.2, LingPipe libraries are used for Chinese segmentation. The library has been trained previously. `L1.text` is segmented and saved as `L1.prep.text`. `L2.text` is tokenised and saved as `L2.prep.text`. `weavefiles.py`, which has been written for Section 4.4.2, is used to format `L1.prep.text` and `L2.prep.text` into `output.snt`. Meanwhile, the mono-lingual file `L1.prep.text` is formatted into `output.snt.1D`.

4.4.9 Translation

This section describes the details of translation step (3) in Section 4.2.1.2. The input files for this step are the formatted Chinese mono-lingual text from Section 4.4.8 (`output.snt.1D`) and the probabilistic multitext grammar model trained by Section 4.4.6 (`model_final`). This step produces the translation output file (`treePerc.tx`), formatted parallel translation text (`treePerc.snt`), as well as interim outputs such as the bilingual parse tree (`treePerc.tb`).

This step is based on the theory of Section 3.3. Translation is achieved by generalised parsing. A special logic is used to load terminal symbols from only one language, while generating bilingual items. The most important configuration files are illustrated in Table 4-2.

Configuration file	Definition	Value
`grammarbuilder.config`	Grammar	`HeadWMTGB` (the bilingual parsing grammar, Section 3.1.3)
`LogicBuilder.config`	Logic	`F` (the translation logic, Section 3.3)

Table 4-2: the GenPar configuration files for translation

The output parse trees are de-integerised and flattened into Chinese and English sentences with several C++ programs (`deinttree`, `appendSpansToTrees` and `linearize`), which are provided by GenPar.

4.4.10 Evaluation

This section describes the details of the evaluation step in Section 4.2.1.2. The input files for this step include the translation output from Section 4.4.9 (`treePerc.tx`) and the formatted test sentences and reference translations from Section 4.4.8 (`L1.prep.text` and `L2.prep.text`). This step produces standard evaluation scores by the Bleu metrics, the NIST metric (`nist.text`), and the F-measure (`FMS1-summary` and `FMS2-summary`).

This step is based on the theory of Section 2.2. The Bleu metrics and NIST measure are produced by making two Python scripts (`evaluate.py` and `bleu.py`) and using a standard Perl script provided by NIST (`nist.pl`). The F-measures are produced by using the Generalised Text Matcher (GTM) tool, which is developed by the natural language processing group of the computer science department of New York University, and provided with GenPar.

4.4.11 Summary

The above sections record the detailed process of a typical experiment, including training, decoding and evaluation. Other work in the experiments that are not mentioned above includes the work of code review (e.g. through reading of the GenPar source code), the compiling of the systems and libraries (e.g. boost C++ library), as well as the development of auxiliary utilities (e.g. an HTTP based script to transfer files between different machines and operating systems).

Details having been recorded, the next section gives an overview of the research questions and the corresponding design of experiments.

4.5 An overview of all the experiments

The first experiment gives the standard evaluation scores of English-Chinese translation by parsing. The accuracy of SMT is expected to be higher after more training. Section 5.1.1 gives comparison to the translation accuracy with different amount of training data.

Besides the amount, the genre of training data is also important to SMT. The difference in the domain and genre between the training and testing data is expected to influence the accuracy. Section 5.1.2 studies the influence of writing style by comparing the accuracy of Mandarin (the simplified Chinese language) and Cantonese (a Chinese dialect) translation.

As introduced in Section 1.1.2.2, grammatical information is expected to be helpful in the development SMT models. One of the potential advantages of SMT by parsing is that this model is based on synchronous grammar trees, which are generated from Chinese and English grammatical information. Section 5.1.3 studies the influence of Chinese and English mono-lingual syntactic information by varying the mono-lingual grammar trees during hierarchical alignment.

Most current SMT systems are based on word-to-word translations, and the accuracy of word-to-word translation probabilities has a general influence on SMT accuracy. Section 5.1.3 studies the influence of word-to-word translation on Chinese-English SMT by parsing by varying the word-to-word probability model during hierarchical alignment.

With observation and analysis of the above experiments, Chapter 5 summarises the advantages and disadvantages of Chinese-English SMT by synchronous parsing. By comparison with other SMT models, it proposes some future work.

Chapter 5

The results and conclusions

5.1 Test results and discussions

5.1.1 Standard tests

5.1.1.1 Standard tests with different training sets

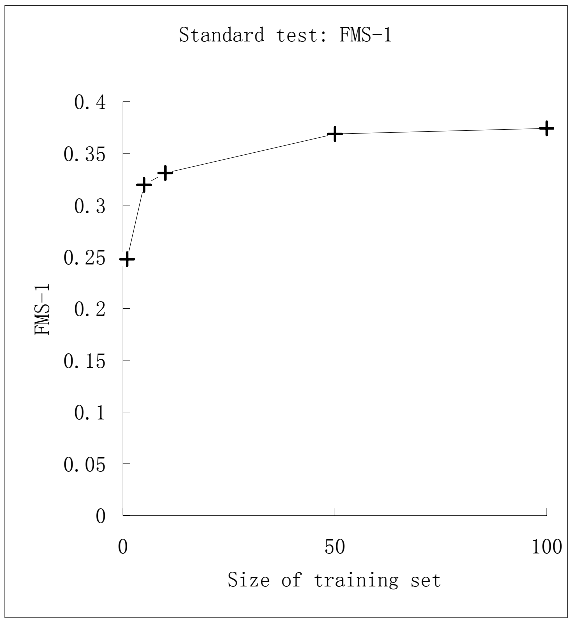
The standard tests show the accuracy and learning curves of the Chinese-English translation experiments. In these tests, five training sets were extracted from the Hong Kong Parallel Text, increasing in size. The same testing set was used, which contains around 1,000 sentences of unseen data extracted from the Hong Kong Parallel Text.

The automatic evaluation scores are shown in Table 5-1. In this table, FMS-*e* stands for the F-measure with weighing factor *e* (Equation 2-12), where *e*=1 (unigram only) and *e*=2 (favouring phrases). BLEU stands for the standard Bleu metrics (Section 2.2.1) and NIST stands for the NIST score (Section 2.2.2).

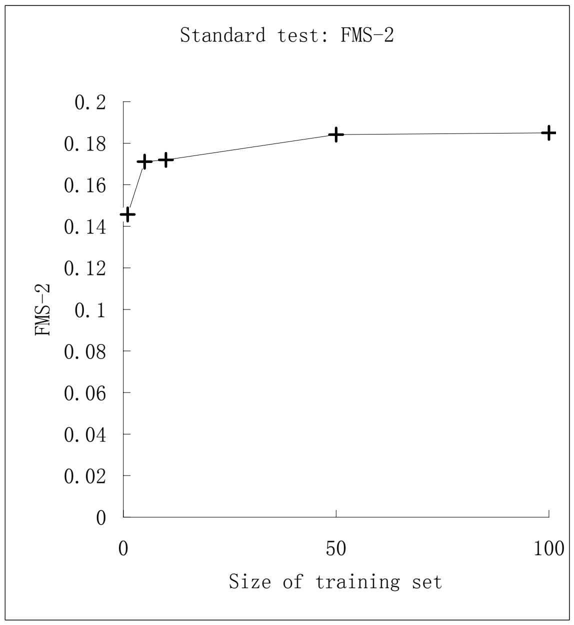
Training set size (sentence pairs)	FMS-1	FMS-2	BLEU	NIST
1k	0.2477	0.1457	0.0155	0.5382
5k	0.3196	0.1711	0.0394	2.0645
10k	0.3310	0.1720	0.0437	2.8120
50k	0.3687	0.1841	0.0604	4.0294
100k	0.3742	0.1850	0.0605	4.1228

Table 5-1: test results for the Chinese-English translation experiment

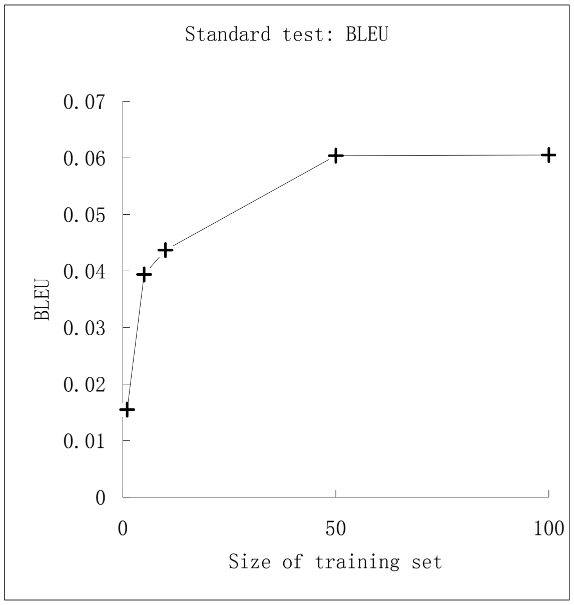
It can be seen that as the training set size increases, all standard scores increase, while the rate of increase falls. The learning curves are illustrated in Figure 5-1.



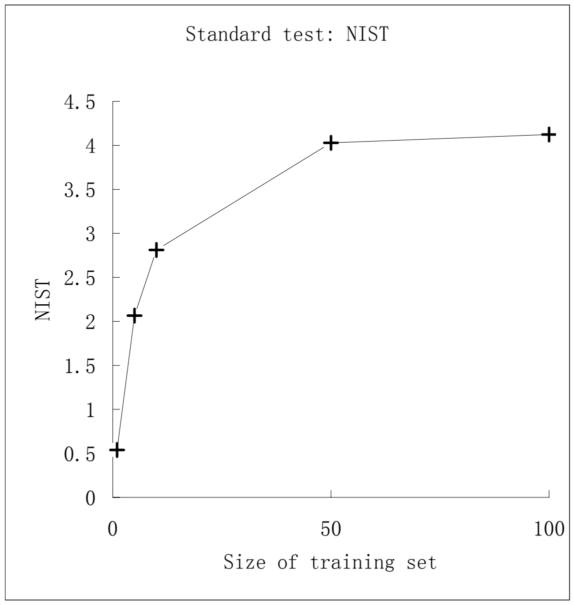
(a) FMS-1



(b) FMS-2



(c) BLEU



(b) NIST

Figure 5-1: the learning curves

5.1.1.2 Example sentences from translation output

The following table contains randomly extracted example translations. For each source sentence, the output translations with different training sets are contrasted.

Original Chinese sentence	training set size	English translation
一九九八年 六月 三日 （ 星期三 ）	1k	一九九八年 六月 3 (Wednesday)
	5k	1998 June 3 (Wednesday)
空气 素质 : 良好	1k	ventillation 素质 : conditions
	5k	air quality : good
行政 长官 会见 美国 国会 代表团	1k	Executive opening meet States 国会 entourage
	5k	Chief Executive addresses States 国会 delegations
	50k	Chief Executive meets US delegation
新 机场 将 命名 为 " 香港 国际 机场 "	1k	new will " Hong international "
	5k	new Airport will 命名为 " Hong international Airport "
	50k	new Airport will " Hong International Airport "
	100k	New Airport will named Hong International Airport "
增加 小学 及 初中 学生 书 簿 津贴 额	1k	additional Profiles and 初中 Students
	5k	increase and students allowances allowance
	10k	increase primary and Junior students objection 簿 allowance applied
	50k	increase primary and junior students CTRs allowances applied
	100k	increase primary and junior students Guidebook allowances applied
警方 呼吁 市民 提供 有关 今 （ 星期三 ） 晨 在 慈云山 发生 的 一 宗 爆窃案 资料	1k	Appeal members of the public Information (Wednesday) in risks a cases information
	5k	Police urged the public information The (Wednesday) 8.08 in occurred a information
	10k	Police urged the public provide The So (Wednesday) 8.08 in 慈云山 occurred 's a cases 爆窃案 information
	50k	Police urged the public provide the this (Wednesday) Tseun at Tsz occurred 's cases burglary information
	100k	Police urged the public provide The this Wednesday morning Tsz occurred fatal burglary information

随着 新 机场 投入 不同 阶段 的 运作 ， 人 手 最终 将 增至 一百五十 人 （ 包括 救护员 在内 ）	1k	Chau-Huanggang new down will (in)
	5k	advent SHWF new Airport different phase Operation persons n't will persons (including arrive in)
	10k	Attached SHWF new Airport entering different 's Operation persons deal Final will persons (including in)
	50k	With hygiene new Airport hoped different Phase 's operated were manpower final will increased 150 were (including ambulance in)
	100k	With Estate New Airport operation different stages 's operations persons hands final will increased 150 persons (including ambulance in in)
发言人 说 ： ″ 任何 人士 如 对 选民 登记 有 任何 疑问 ， 可 致电 选举 事务处 热线 电话 ， 号码 为 二八九一 一零零一 ″	1k	spokesman said : “ Anyone contact Anyone can call classes hotline Enquiry is . ”
	5k	spokesman said : “ Anyone For External elector job-seekers have any asked can call Electoral Office hotline Tel. is 2891 . ”
	10k	spokesman said : “ Anyone are follows voters registered has any asked can call Electoral Office hotline Tel numbers for addressed .
	50k	spokesman said “ Anyone are For on Voter registration are any doubt that can call Electoral Office telephone hotline that numbers for 1001 .
	100k	spokesman said Any person For to Voter registration further any doubt that can call Electoral Office hotline Tel number for 1001 .

Table 5-2: sentences extracted from the translation output

It can be seen that generally the translation becomes more accurate with more training data, although there are exceptions with some sentences. Also, the translations are generally much less accurate compared to human translators. Moreover, for longer sentences, the grammatical structures are comparatively less accurate.

5.1.1.3 Comparison with existing experiments of translation by parsing

This Section shows the results of existing experiments done on SMT by parsing. Table 5-3 shows the results of Arabic-English translation by parsing, while Table 5-4 shows the results of French-English translation by parsing.

Training set size	FMS-1	FMS-2	BLEU	NIST
7k	0.1720	0.0831	0.0151	1.7606

Table 5-3: existing test result of Arabic-English translation by GenPar, adapted from (Burbank et al., 2005). The only training set has 7,000 sentences.

Training set size	FMS-1	FMS-2	BLEU	NIST
5k	0.2824	0.1170	0.0409	2.6369
10k	0.2996	0.1217	0.0499	2.8515
50k	0.3247	0.1270	0.0625	3.0895
100k	0.3639	0.1332	0.0799	3.3484

Table 5-4: existing test result of French-English translation by GenPar, adapted from (Burbank et al., 2005).

From the above tables, it can be seen that the accuracy of Chinese-English translation by parsing of this thesis is comparable to the accuracy of existing translation by parsing experiments.

5.1.2 The influence of the writing style

5.1.2.1 Test summary

Apart from the size, the content of training data is also important to the accuracy. In order to test the influence of different writing styles, two test sets are used in the experiments. The first test set contains around 1,000 unseen sentences from the Hong Kong Parallel Text, which has been written in Cantonese. The second test set contains around 1,000 unseen sentences from the MTC Corpus Part 3, which has been written in Mandarin. The training set is in Cantonese.

Table 5-5 shows the test results. Test set HK stands for the Cantonese test set from Hong Kong Parallel Text, while test set MTC stands for the Mandarin test set from Multiple-Translation Chinese.

Training set size	Test set	FMS-1	FMS-2	BLEU	NIST
1k	HK	0.2477	0.1457	0.0155	0.5382
	MTC	0.1599	0.0861	0.0070	0.0319
5k	HK	0.3196	0.1711	0.0394	2.0645
	MTC	0.2095	0.0984	0.0132	0.2667
10k	HK	0.3310	0.1720	0.0437	2.8120
	MTC	0.2278	0.1016	0.0184	0.7242
50k	HK	0.3687	0.1841	0.0604	4.0294
	MTC	0.2672	0.1042	0.0179	2.1368
100k	HK	0.3742	0.1850	0.0605	4.1228
	MTC	0.2794	0.1072	0.0181	2.3334

Table 5-5: test results for the influence of writing styles. The training set is in Cantonese, while the two test sets are in Cantonese and Mandarin, respectively.

The scores from the first F-measure (FMS-1) are illustrated in the figure below.

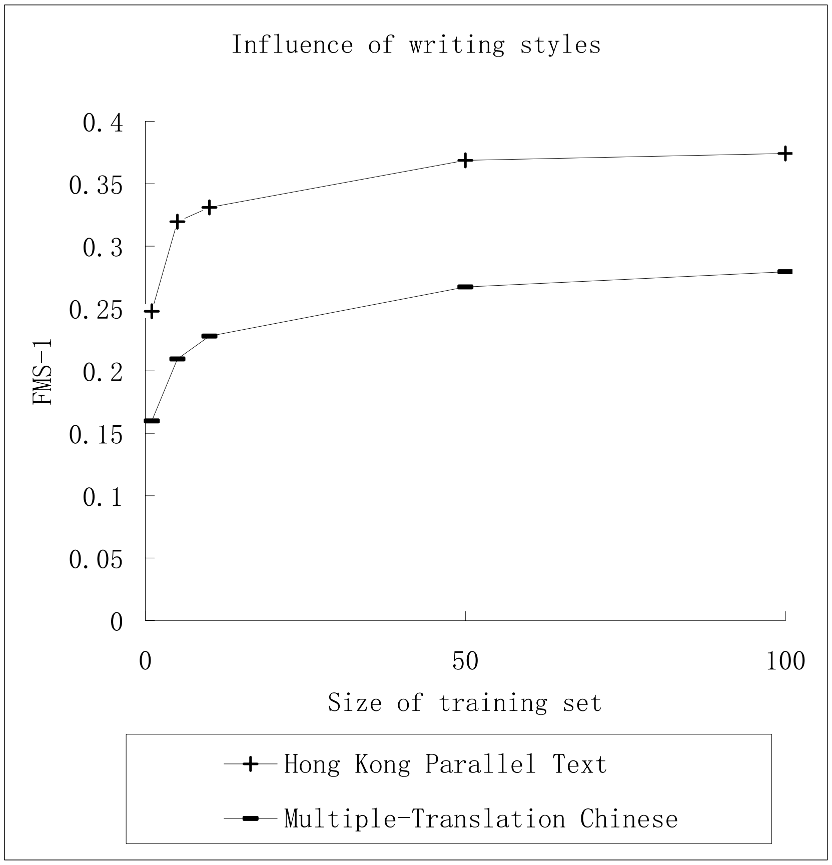


Figure 5-2: influence of writing styles

Figure 5-2 shows that as the training set size increases, the FMS-1 accuracy of the MTC test increases in a curve that is similar to the HK test. However, it is always smaller than the HK test.

5.1.2.2 Discussion

It can be seen from the test results that writing style has an influence on translation accuracy. The most import reason may be that the bilingual grammar model, which is trained on one style of sentences, does not analyse another style as accurately. The use of Chinese words is comparatively flexible, and each word can take many types of part-of-speech. This makes the influence of writing style more obvious.

Another reason may be the difference in the use of words. For example, in Hong Kong Parallel Text, "air quality" is expressed as "空气 素质", while the same phrase in the Multiple-Translation Chinese is expressed as "空气 质量". Such cases of different expressions are not rare. Another example may be "computer program", in which "program" is translated into "程式" in Cantonese but "程序" in Mandarin.

Writing style has a general influence on SMT. This influence is apparent between different domains and subjects. For example, manual tests on mainstream public translation systems (e.g. Google Translation) shows that the translation between English and Chinese performs comparatively better with news articles than with essays on information technology. This is probably because these systems are trained more frequently on general news articles.

5.1.3 The role of syntactic information

As introduced in the end of Section 1.1.2.2, being high-level human abstractions of languages, syntax is expected to help improving the SMT models. A potential advantage of SMT by parsing is the use of syntactic information. This experiment tests the influence of syntactic information. However, in the absence of manually parsed bilingual treebanks, it is impossible to test the influence of bilingual GMTG directly. Instead, this experiment tests the influence of its components in hierarchical alignment – Chinese and English mono-lingual syntactic information.

In both the following experiments, the source bilingual texts for the training set are 10,000 sentence pairs from Hong Kong Parallel Text. Meanwhile, different amounts of mono-lingual syntactic information are used for hierarchical alignment, from which the bilingual treebank for GMTG training is generated. The rest of the process is the same as a standard test.

5.1.3.1 The influence of Chinese (source language) syntactic information

This section tests the influence of Chinese syntactic information by comparing two different methods for hierarchical alignment – one with Chinese syntactic information (Section 3.2.2) and the other without it (Section 3.2.1). The comparison is illustrated in Table 5-6.

Chinese syntactic information	FMS-1	FMS-2	BLEU	NIST
Used	0.3310	0.1720	0.0437	2.8120
Not used	0.3384	0.1773	0.0471	2.7091

Table 5-6: the influence of Chinese (source language) syntactic information

It can be seen from the table that the standard scores are comparable in the two test cases. No significant influence of Chinese syntactic information is observed.

5.1.3.2 The influence of English (target language) syntactic information

This section tests the influence of English syntactic information by using a different method from the last section. In this experiment, different accuracies of English syntactic information are used to produce the bilingual treebank, and the effect is compared. This is done by varying the size of English training data for the Bikel parser before producing the mono-lingual treebanks with it. Table 5-7 illustrates the comparison between English training sets of three different sizes.

Training set size for Bikel parser	FMS-1	FMS-2	BLEU	NIST
1k	0.3338	0.1731	0.0442	2.9169
10k	0.3282	0.1712	0.0414	2.7526
50k	0.3310	0.1720	0.0437	2.8120

Table 5-7: the influence of English (target language) syntactic information

It can be seen from the table that the standard scores are comparable in the three test cases. No significant influence of English syntactic information is observed.

5.1.3.3 Conclusion

In the above experiments, neither English nor Chinese mono-lingual syntactic information has a significant influence on the standard evaluation scores.

There are two possible reasons for this. Firstly, the accuracy of GMTG may have no significant impact on translation. That is to say, syntactic information does not play an important role in Chinese-English SMT by GMTG parsing. In comparison, Och et al. (2004) examined the influence of various features on a maximum entropy SMT model, and found no significant influence of syntactic information. However, recent research by Quirk and Corston-Oliver (2006) shows the influence of English syntactic information on a different tree-based SMT model, which combines phrase-based information with syntactic treelets. Therefore, it may be concluded that the role of syntactic information on SMT is dependent on the probabilistic model, and with probabilistic GMTG, syntax information is comparatively less influential. However, the above conclusion can only be confirmed with the availability of confidently accurate (i.e. manually parsed) bilingual treebanks.

The second possible reason for the experiment result is the noise in hierarchical alignment, which is currently a necessary step to derive bilingual treebanks. This noise may come from the inaccuracy of mono-lingual parsers, the word alignment algorithm, as well as the hierarchical alignment process. What is more, sentence pairs from the Hong Kong Parallel Text are correspondent in meaning, instead of CFG structures. By observation, there are over 20% sentence pairs that are not syntactically correspondent to each other. The noise in the resulting bilingual treebank can be significant enough to outweigh the influence of mono-lingual syntactic information.

5.1.4 The importance of word-to-word probabilities

5.1.4.1 Test summary

Most current SMT systems are based on word-to-word translations. In the experiments of Chinese-English SMT by parsing, word-to-word translation probabilities are a basis for hierarchical alignment. This experiment tests the influence of word-to-word translations.

The following experiments use different amounts of sentence pairs from the Hong Kong Parallel Text for word alignment. Meanwhile, they use 10,000 sentence pairs from Hong Kong Parallel Text for the other training steps. The rest of the process is the same as the standard test.

The test results are illustrated in Table 5-8.

Size of word alignment data	FMS-1	FMS-2	BLEU	NIST
10k	0.3310	0.1720	0.0437	2.8120
50k	0.3560	0.1758	0.0500	3.6036
100k	0.3596	0.1772	0.0520	3.5906

Table 5-8: the influence of word alignment

It can be seen from the table that as more sentence pairs are used for word alignment, the automatic evaluation scores generally increase.

5.1.4.2 Discussion

The word alignment algorithm is based on the EM iteration in Section 2.1.2. When more sentence pairs are used, the output word alignment model should include more words, while the accuracy of the model increases. The experiment shows that the translation accuracy increases with more word alignment data. This is mainly because the accuracy of word-to-word probabilities is a crucial starting point for the bottom-up hierarchical alignment algorithm (Section 3.2).

It should be noticed that the rate of increase of the evaluation score drops as more sentence pairs are used for word alignment. One of the reasons is that hierarchical alignment only makes use of words that appear in its training sentence pairs, which is fixed to 10,000 in size. Thus the accuracy of word alignment within the 10,000 sentence pairs is more important to translation than the total number of words included in the alignment model. The above reasoning can also be an explanation of the exceptions in Table 5-8, such as the NIST score with 100k sentence pairs.

Word-to-word translation is generally crucial to SMT. From the human translation’s point of view, words and unbreakable phrases are also the basic elements to be translated.

Finally, it should be noted that SMT by GMTG parsing is based on words instead of phrases – phrases are not considered separately as terminal symbols in GMTG, but integrated into the production rules. This is a potential disadvantage to the model. As a result, SMT by GMTG parsing may also have the problem of word-based models discussed in Section 2.1.3. What is more, the word based alignment may result in lower Bleu scores than phrase-based models, since the Bleu evaluation is based on n-grams. This is less important, however, as long as the translation is reasonable by the human evaluation.

5.2 Summary

Like human translation, machine translation has two essential factors – unit element (unbreakable word or phrase that carries meaning) translation and target sentence organisation. The simplest models for SMT are word-based, where the unit elements are words and sentence organisation is modelled by comparatively simple mechanisms such as word reordering. One of the main improvements of phrase-based models over the word-based models is on the definition of unit elements, which includes phrases. Hierarchical phrase based and tree-based models further improved the target sentence organisation. The models have improved translation accuracy by evolving towards a higher level of abstraction, while word alignment often serves as the basis for more complex models.

SMT by parsing can be seen as using a tree-based model. It also makes use of syntactic information, which has been helpful to SMT systems in modelling hierarchical constituent movement and target language generation. The decoding process for SMT by parsing is special in that the translation sentence is generated as a by-product of parsing with bilingual synchronous grammars, which requires recursive correspondence between the grammatical structures of the two languages. The advantages of this decoding process include the reuse of existing parsing algorithms to save engineering cost.

Chinese-English SMT is a special instance of statistical machine translation. While it benefits from general SMT methods, it is also influenced by the specific properties of both languages. Based on the GenPar system and other pieces of software, this thesis studies Chinese-English SMT by bilingual GMTG parsing with large-scale experiments. The results show that the accuracy of Chinese-English translation by parsing is comparable with existing French-English and Arabic-English translations by GMTG parsing from the literature. What is more, although word-alignment has significant influence on translation, mono-lingual syntactic Chinese and English information is not as influential in the experiments.

The experiments also reveal that SMT systems have much to improve. For example, the overall accuracy is still far below human translation, and the effect of translation is considerably influenced by writing style and topic. From the above experiments, several possible further improvements can be suggested. They are discussed in the next section.

5.3 Future work

This section presents possible improvements to SMT models. The possible future works are organised by the two main factors summarised in the last section – unit element translation and sentence organisation.

The word-to-word and phrase-to-phrase alignment models are crucial to any SMT system. Because words and phrases are the atomic elements that carry the meaning of a sentence, it is likely that a translation is understandable with words and phrases correctly translated, even if the order is inaccurate. Currently most SMT systems use EM based learning methods, while there are Chinese-English dictionaries available without probabilities. A possible improvement is making use of off-the-shelf dictionaries during the training of word and phrase alignments.

Another observation from the experiments is that word translations can be dependent on context. A word can have many different translations, while for a certain source sentence there is a best choice. Thus it is reasonable to consider a word-to-word model that includes contextual information explicitly. The difficulty is the choice of context. When grammar is considered, an example context may be the head word.

Sentence organisation can be improved by better models, while syntactic information is an important factor. In the experiments of Chinese-English SMT by synchronous GMTG parsing, mono-lingual syntactic information did not prove to have a significant influence. Apart from the noise introduced by the hierarchical alignment, some further thoughts arise about the model itself:

Firstly, there is the possibility that the accuracy of bilingual GMTG (or combined CFG) does not have a significant influence. The above possibility can only be demonstrated when the corresponding Chinese-English bilingual treebank is available in a reasonable size in the future. Meanwhile, it is meaningful to think of potentially better representation of the correspondence between Chinese and English sentences. As was seen from the Hong Kong Parallel Text, a significant amount of parallel sentences are correspondent in meaning, but not in CFG syntactic structures. Therefore, the candidates which express more common structural alignment between translation pairs may include more semantic-prone grammar structures, such as the *lexical functional grammar* (LFG; Dalrymple, 2001) and *combinatory categorical grammar* (CCG; Steedman, 2000).

Secondly, potentially better models may also use no hierarchical structural combination. On the one hand, the same meaning can be best carried out by completely different syntactic structures in Chinese and English. On the other hand, mono-lingual syntactic information is clearly helpful in the generation of sentences. Therefore, it may be useful to consider a two-phase decoding model that does not include any recursive structural correspondence, but separates source sentence analysis and target sentence generation. In this way the target language grammar can be used specifically for generating translations. What is more, grammatical information for only one language, such as the tense of English sentences, can be explicitly modelled. The difficulty is the intermediate representation, and the disadvantage of this model is that existing techniques in generalised parsing may be inapplicable for reuse.

References

A. Aho and J. Ullman. 1969. *Syntax directed translations and the pushdown assembler*. Journal of Computer and System Sciences, 3: 37–56, February.

James Allen. 1995. *Natural Language Understanding (2nd edition)*. Addison-Wesley.

Yaser Al-Onaizan, Jan Curin, Michael Jahr, Kevin Knight, John Lafferty, Dan Melamed, Franz-Josef Och, David Purdy, Noah A. Smith, David Yarowsky. 1999. *Statistical Machine Translation Final Report*. JHU Workshop 1999.

James Baker. 1979. *Trainable grammars for speech recognition*. In Speech Communications Papers for the 97th Meeting of the Acoustical Society of America, pp. 547–550, Boston, MA, June.

Daniel Bikel. 2002. *Design of a Multi-lingual, Parallel-processing Statistical Parsing Engine*. In Human Language Technology Conference (HLT), San Diego.

A. Burbank, M. Carpuat, S. Clark, M. Dreyer, P. Fox, D. Groves, K. Hall, M. Hearne, D. Melamed, Y. Shen, A. Way, B. Wellington, and D. Wu. 2005. *Final Report of the 2005 Language Engineering Workshop on Statistical Machine Translation by Parsing*. JHU Workshop 2005.

Brooke Cowan, Ivona Kucerova, and Michael Collins. 2006. *A Discriminative Model for Tree-to-Tree Translation*. To appear in proceedings of the Conference on Empirical Methods in Natural Language Processing 2006.

Peter F. Brown, Vincent J. Della Pietra, Peter V. deSouza, Jenifer C. Lai, Robert L. Mercer. 1990. *Class-Based N-Gram Models of Natural Language*. Computational Linguistics, 16(2): 79–85.

Peter F. Brown, Stephen A. Della Pietra, Vincent J.Della Pietra, Robert. L. Mercer. 1993. *The mathematics of statistical machine translation: Parameter estimation*. Computational Linguistics, 19(2): 263–311.

Eugene Charniak. 2000. *A maximum-entropy-inspired parser*. In Proceedings of North American ACL (NAACL), pp, 132–139.

David Chiang. 2005. *A Hierarchical Phrase-Based Model for Statistical Machine Translation*. In Proceedings of ACL-05, pp. 263–270.

Noam Chomsky. 1956. *Three models for the description of language*. I.R.E. Transactions on Information Theory, vol. IT-2, no. 3: 113–24.

John Cocke and Jacob T. Schwartz. 1970. *Programming languages and their compilers: Preliminary notes*. Technical report, Courant Institute of

Mathematical Sciences, New York University, 1970.22.

Michael Collins. 1996. *A New Statistical Parser Based on Bigram Lexical Dependencies*. In Proceedings of ACL-96, pp. 184–191.

Michael Collins. 1997. *Three Generative, Lexicalised Models for Statistical Parsing*. In Proceedings of the ACL-97, pp. 16–23, Madrid, Spain, July.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2001. *Introduction to Algorithms, Secion Edition*. MIT Press & McGraw-Hill.

Mary Dalrymple. 2001. *Lexical Functional Grammar*. New York: Academic Press.

George Doddington. 2002. *Automatic evaluation of machine translation quality using n-gram cooccurrence statistics*. ARPA Workshop on Human Language Technology.

Jay Earley. 1970. *An efficient context-free parsing algorithm*. Communications of the Association for Computing Machinery, 13(2): 94–102.

Roger Garside, Geoffrey Leech and Geoffrey Sampson. 1987. *The Computational Analysis of English: A Corpus-based Approach*. London: Longman.

Daniel Gildea. 2003. *Loosely Tree-Based Alignment for Machine Translation*. In Proceedings of ACL-03, pp. 80–87, Sapporo, Japan.

Joshua Goodman. 1998. *Parsing Inside-out*. Ph.D. Thesis, Harvard University.

John Hutchins. 2005. The history of machine translation in a nutshell. http://ourworld.compuserve.com/homepages/wjhutchins/Nutshell.htm

Frederick Jelinek. 1969. *A fast sequential decoding algorithm using a stack*. IBM Journal of Research and Development, vol. 13, pp. 675–685, 1969.

Daniel Jurafsky and James H. Martin. 2000. *An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice-Hall.

T. Kasami. 1965. *An efficient recognition and syntax-analysis algorithm for context-free languages*. Technical report AFCRL-65758, Air Force Cambridge Research Laboratory.

Kevin Knight, Daniel Marcu. 2004. *Machine Translation in the Year 2004*. In Intl. Conf. on Acoustics, Speech, and Signal Processing (ICASSP), 965–968.

Philipp Koehn, Franz Josef Och, Daniel Marcu. 2003. *Statistical Phrase-Based Translation*. In Proceedings of HLT, pp. 127–133.

Karim Lari, Steve J. Young. 1990. *The estimation of stochastic context-free grammars using the inside-outside algorithm*. Computer Speech and Language, 4: 35–56, 1990.

Robert Lowth. 1762. *A Short Introduction to English Grammar*. Scholars Facsimilies & Reprint.

David Magerman. 1995. *Parsing as Statistical Pattern Recognition*. Ph.D. Thesis, Stanford University.

David McClosky, Eugene Charniak, Mark Johnson. 2006. *Effective Self-Training for Parsing*. In Proceedings of North American ACL (NAACL).

I. Dan Melamed. 2003. *Multitext Grammars and Synchronous Parsers*. In Proceedings of the 2003 Meeting of the North American chapter of the Association for Computational Linguistics (NAACL-03), Edmonton.

I. Dan Melamed, Giorgio Satta and Benjamin Wellington. 2004. *Generalized multitext grammars*. In Proceedings of ACL-04, Barcelona, Spain.

I. Dan Melamed, Wei Wang. 2005. *Statistical Machine Translation by Generalized Parsing*. Proteus Project Working Paper #2.

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

Franz Josef Och. 1999. *An Efficient Method for Determining Bilingual Word Classes*. In Proceedings of EACL-99, Bergen, Norway.

Franz Josef Och, Christoph Tillman and Hermann Ney. 1999. *Improved alignment models for statistical machine translation*. In Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (in conjunction with ACL-99), pp. 20–28, College Park, MD, USA, June.

Franz Josef Och, Hermann Ney. 2000. *Improved Statistical Alignment Models*. In Proceedings of ACL-2000, pp. 440–447, Hong Kong, China, October.

Franz Josef Och and Hermann Ney. 2002. *Discriminative Training and Maximum Entropy Models for Statistical Machine Translation*. In Proceedings of Annual Meeting of the Association for Computational Linguistics, pp. 295–302, Philadelphia, PA, July.

Franz Josef Och, Hermann Ney. 2003. *A Systematic Comparison of Various Statistical Alignment Models*. Computational Linguistics, 29(1): 19–51.

Franz Josef Och, Daniel Gildea, Sanjeev Khudanpur, Anoop Sarkar, Kenji Yamada, Alex Fraser, Shankar Kumar, Libin Shen, David Smith, Katherine Eng, Viren Jain, Zhen Jin and Dragomir Radev. 2004. *A Smorgasbord of Features for Statistical Machine Translation*. In Proceedings of HLT/NAACL 2004, Boston.

Kishore Papineni, Salim Roukos, Todd Ward, Wei-Jing Zhu. 2001. *BLEU: A Method for Automatic Evaluation of Machine Translation*. In Proceedings of ACL-02, pp. 311–318, Philadelphia, PA, July.

Chris Quirk and Simon Corston-Oliver. 2006. *The impact of parse quality on syntactically-informed statistical machine translation*. To appear in proceedings of the Conference on Empirical Methods in Natural Language Processing 2006.

Adwait Ratnaparkhi. 1997. *A simple introduction to maximum entropy models for natural language processing*. IRCS Report 97-08, University of Pennsylvania, 3401 Walnut Street, Suite 400A, Philadelphia, PA, May.

Azriel Rosenfeld. 1968. *An Introduction to Algebraic Structures*. New York, Holden-Day.

Stuart Russell and Peter Norvig. 2003. *Artificial Intelligence: A Modern Approach (Second Edition)*. Prentice Hall.

Geoffrey Sampson. 1986. *Simulated Annealing as a Parsing Technique*. In University of Leeds Working Papers in Linguistics and Phonetics 4: 43–60.

Mark Steedman. 2000. *The Syntactic Process.* MIT Press.

Chrostoph Tillmann, Stephan Vogel, Hermann Ney and Alex Zubiaga. 1997. *A DPbased Search Using Monotone Alignments in Statistical Translation*. In Proceedings of ACL/EACL-97, pp. 289–296, Madrid, Spain, July.

Kristina Toutanova and Christopher D. Manning. 2000. *Enriching the Knowledge Sources Used in a Maximum Entropy Part-of-Speech Tagger*. In Proceedings of the Conference on Empirical Methods in Natural Language Processing.

Joseph Turian, Luke Shen, I. Dan Melamed. 2003. *Evaluation of Machine Translation and its Evaluation*. In Proceedings of MT Summit IX, pp. 386–393, New Orleans, LA, September.

Stephan Vogel, Hermann Ney, Christoph Tillmann. 1996. *HMM-based word alignment in statistical translation*. In Proceedings of ACL-96, pp. 836–841, Copenhagen, Denmark, August.

Dekai Wu. 1997. *Stochastic Inversion Transduction Grammars and Bilingual Parsing of Parallel Corpora*. Computational Linguistics, 23(3): 377–404.

Kenji Yamada and Kevin Knight. 2001. *A syntax-based statistical translation model*. In Proceedings of ACL-01, pp. 523–529, Toulouse, France.

Daniel H. Younger. 1967. *Recognition and parsing of context-free languages in time n^3*. Information and Control, 10(2): 189–208.

Sample code

1. bleu.py – implementing the Bleu metrics (Section 2.2.1)

```
g_sWelcome = """
Bleu.py - the bleu metrics for machine translation.
Yue Zhang @ Oxford. 2006.
"""
g_sUsage = "Bleu.py [-d] [-c] [-v] [--N=n-gram] \
candidate reference1 [reference2, ...]"

#==================================================
#
# Import
#
#==================================================

import sys
import getopt
import math

#==================================================
#
# The global variables
#
#==================================================

N = 4                           # N for N-Gram
NGramsCandidate = []            # candidate ngrams
NGramsMatch = []                # reference ngrams
lengthCandidate = 0             # candidate length
lengthReference = 0             # reference length

g_Debug = 0                     # debug switch
g_Verbose = 0                   # verbose switch
g_CaseSensitive = 0             # case sensitive/n

#--------------------------------------------------
#
# readOneLine - read one line from a candidate
#               file and a list of reference files.
#
# Input: fileCandidate - the candidate file
#        fileReferences - the reference files
#
# Return: list of lines
#
#--------------------------------------------------

def readOneLine(fileCandidate, fileReferences):
   lData = [] # the return value
   # Read one line from candidate.
   sLine = fileCandidate.readline()
   if not sLine:
      return None
   if g_CaseSensitive:
      lData.append(sLine.strip())
   else:
      lData.append(sLine.strip().lower())
   # Read one line from each reference file.
   for fileReference in fileReferences:
      sLine = fileReference.readline()
      if not sLine:
         return None
      if g_CaseSensitive:
         lData.append(sLine.strip())
      else:
         lData.append(sLine.strip().lower())
   return lData

#--------------------------------------------------
#
# processOneLine - process one line
#
#--------------------------------------------------

def processOneLine(lLines):
   global lengthCandidate, lengthReference
   sCandidateLine = lLines[0]
   lWords = sCandidateLine.split(" ")
   # Count each N-Gram.
   for n in xrange(1, N+1):
      setCandidateNGrams = set([])
      for m in xrange(0, len(lWords)+1-n):
         setCandidateNGrams.add(\
                           " ".join(lWords[m:m+n]))
      for sNGram in setCandidateNGrams:
         countCandidate = nGramCount(\
                              sCandidateLine, sNGram)
         countReference = 0
         for sReferenceLine in lLines[1:]:
            countReference = max(countReference, \
               nGramCount(sReferenceLine, sNGram))
         NGramsMatch[n-1] += min(countReference, \
                                    countCandidate)
         NGramsCandidate[n-1] += countCandidate
   # Count length.
   lengthCandidate += len(lWords)
   lengthTemp = lLines[1].count(" ") + 1
   for sReferenceLine in lLines[2:]:
      lengthTemp = min(lengthTemp, \
                  sReferenceLine.count(" ") + 1)
   lengthReference += lengthTemp

#--------------------------------------------------
#
# nGramCount - count n-grams from string
#
# Input: sString - the string
#        sNGram - the n-gram
#
# Return: count
#
#--------------------------------------------------

def nGramCount(sString, sNGram):
   if g_Verbose:
      print "N-gram count for %s in %s is %d" % \
         (sNGram, sString, sString.count(sNGram))
   return sString.count(sNGram)
```

```
#-------------------------------------------------
#
# computeBleu - compute the bleu score
#
# Return: bleu
#
#-------------------------------------------------

def computeBleu():
   # List of N-Gram precisions.
   precision = []
   for n in xrange(0, N):
      if NGramsMatch[n] == 0:
         precision.append(1)
      else:
         precision.append(float(NGramsMatch[n]) \
                                     /NGramsCandidate[n])
   # The precision.
   fPrecision = 0
   for n in xrange(0, N):
      fPrecision += math.log(precision[n])/N
   fPrecision = math.exp(fPrecision)
   if g_Debug:
      print "Modified n-gram precision is:", \
                                          fPrecision
   # The brevity penalty score.
   BP = 1
   if (lengthReference > lengthCandidate):
      BP = math.exp(1 - \
            float(lengthReference)/lengthCandidate)
   if g_Debug:
      print "Brevity penalty is:", BP
   # The Bleu score here.
   return fPrecision * BP
```

```
#=================================================
#
# The main entry.
#
#=================================================

if __name__ == '__main__':
   # Show welcome.
   print g_sWelcome
   # Get system arguments.
   optlist, args = getopt.getopt(sys.argv[1:], \
                                  "dvc", ["N="])
   if len(args) < 2:
      print g_sUsage
      sys.exit(1)
   for opt, val in optlist:
      if opt == "-d":
         g_Debug = 1
      elif opt == "-v":
         g_Verbose = 1
      elif opt == "-c":
         g_CaseSensitive = 1
      elif opt == "--N":
         N = int(val)
   for n in xrange(0, N):
      NGramsCandidate.append(0)
      NGramsMatch.append(0)
   # Get candidate and reference files.
   fileCandidate = open(args[0])
   fileReferences = []
   for sReference in args[1:]:
      fileReferences.append(open(sReference))
   # Analyse sentences.
   lLines = readOneLine(fileCandidate, \
                                  fileReferences)
   while lLines:
      processOneLine(lLines)
      lLines = readOneLine(fileCandidate, \
                                  fileReferences)
   # Print the Bleu score.
   print computeBleu()
   # Close candidate and reference files.
   fileCandidate.close()
   for fileReference in fileReferences:
      fileReference.close()
```

2. weavefiles.py – the module to join lines together (Section 4.4.2.2)

```
#******************************************************************
#
# weavefiles.py - A file joining script.
#
# Provide a list of files to join as the system arguments.
# Output to the standard system output device by default.
#
# Yue Zhang
# Comlab, Oxford.
# 2006
#
#******************************************************************

#==================================================================
#
# Global settings - which symbols are used to group corresponding lines
#
#==================================================================

g_lInput = []
g_sStart = '<s>\n'
g_sEnd = '</s>'

#------------------------------------------------------------------
#
# weaveFiles - weave the input files with designated start and end symbols
#
#------------------------------------------------------------------

def weaveFiles(lInputNames, sStartSt = "", sEndSt = ""):
   # Open files.
   lInput = []
   for sInput in lInputNames:
      lInput.append(open(sInput, "r"))
   # Weave lines.
   bRunning = True
   while bRunning:
      lLine = ["%s" % sStartSt]
      for oInput in lInput:
         sLine = oInput.readline()
         if sLine:
            lLine.append(sLine)
         else:
            bRunning = False
      if bRunning:
         lLine.append("%s" % sEndSt)
         print "".join(lLine)
   # Close files.
   for oInput in lInput:
      oInput.close()

#==================================================================
#
# The main methods
#
#==================================================================

if __name__ == "__main__":
   import sys
   if len(sys.argv) > 1:
      for nIndex in xrange(1, len(sys.argv)):
         g_lInput.append(sys.argv[nIndex])
   weaveFiles(g_lInput, g_sStart, g_sEnd)
```