Chapter 2
BTG-Based SMT

Abstract This chapter systematically introduces *BTG-based SMT*. We first introduce the Bracketing Transduction Grammar and a unified framework for BTG-based SMT, including the model and decoding algorithm that does not integrate language model. We then present an algorithm to integrate standard *n*-gram language models into the decoder. Following that, we describe two extensions to these traditional language models: a backward language model that augments the conventional forward language model, and a mutual information trigger model which captures long-distance dependencies that go beyond the scope of standard *n*-gram language models. By these two models, we attempt to enhance the ability of conventional *n*-gram language models in capturing richer contexts and long-distance dependencies. Finally, we thoroughly compare BTG-based SMT with other SMT formalisms, such as (hierarchical) phrase-based and linguistically syntax-based SMT, so that we can clearly understand the strengths and weaknesses of BTG-based SMT.

BTG-based SMT is one of state-of-the-art SMT formalisms, which is comparable to hierarchical phrase-based and syntax-based SMT in terms of BLEU-measured translation quality (He et al. 2008). BTG-based SMT possesses the following characteristics.

- It is capable of long-distance and hierarchical reordering.
- It is built upon the minimum case of synchronous context-free grammars, which avoids extracting a large number of rarely used translation rules.
- It is both phrase-based and formally syntax-based SMT. It is phrase-based SMT because it uses phrases as translation units, while formally syntax-based SMT in that it constructs hierarchal structures during translation. From this perspective, BTG-based SMT is a natural bridge that connects both phrase-based and syntax-based SMT.

Because of these properties of BTG-based SMT, we select it as the base platform to discuss linguistically motivated SMT in some chapters.

D. Xiong and M. Zhang, *Linguistically Motivated Statistical Machine Translation*, DOI 10.1007/978-981-287-356-9_2

This chapter serves the purpose of systematically introducing BTG-based SMT. The remainder of this chapter proceeds as follows.

- Section 2.1 provides a gentle introduction of Bracketing Transduction Grammar.
- Section 2.2 describes a unified framework for BTG-based SMT, including the whole log-linear model, CKY-style decoding algorithm, and reordering model.
- Section 2.3 elaborates the n-gram language model integration algorithm.
- Section 2.4 presents two extensions to the standard n-gram language model: (1) a backward language model that augments a conventional forward n-gram language model with succeeding words and (2) a trigger language model that captures long-distance dependencies that go beyond the scope of the standard n-gram language model. We give details of these two extensions on modeling, training procedure, and decoding integration.
- Section 2.5 introduces two threshold pruning methods that speed up the CKY-style decoder of BTG-based SMT.
- In order to highlight the strengths and weaknesses of BTG-based SMT, we compare it with other SMT formalisms from various perspectives in Sect. 2.6.
- Finally, we summarize the chapter in Sect. 2.7 and provide additional readings.

2.1 Bracketing Transduction Grammar

The normal form of Bracketing Transduction Grammar is first proposed by Wu (1996) for word-based machine translation. It is formulated as follows:

$$\begin{aligned} X &\rightarrow [X_1, X_2] \\ X &\rightarrow \langle X_1, X_2 \rangle \\ X &\rightarrow e/f \\ X &\rightarrow \epsilon/f \\ X &\rightarrow e/\epsilon \end{aligned} \tag{2.1}$$

The first two productions are *bracketing rules* which combine two neighboring items into a larger item in a *straight* or *inverted* order. Here we use "[]" to denote a straight order and "⟨⟩" an inverted order. The two bracketing rules can be also written as

$$\begin{aligned} X &\rightarrow X_1 X_2 / X_1 X_2 \\ X &\rightarrow X_1 X_2 / X_2 X_2 \end{aligned}$$

This clearly explains the meaning of the straight/inverted orientation. In the straight order, the first nonterminal on the source side is aligned to the first nonterminal on the target side, the second to the second. In other words, the source and target side have the same word order. In the inverted orientation, however, the first nonterminal on the source side is aligned to the second on the target side and the second to the first. This means that on the target side, the word order is completely reversed.

The last three productions are *lexical rules* which respectively translate a source word f to a target word e, f to the null word ε and the null word to the target word e.

BTG is a simplified version of Inversion Transduction Grammar (ITG) (Wu 1997) because it only uses one single undifferentiated nonterminal. As it contains only one nonterminal and five productions, BTG can also be considered as the minimal case of synchronous context-free grammars (SCFG[1]) which can be used for machine translation or bilingual parsing. Even so, this grammar is able to cover the full range of reorderings generated by any ITG (Wu 1997). Furthermore, BTG can also model long-distance reorderings with a tractable polynomial time complexity (Wu 1996). This is the most important advantage that motivates the use of BTG in machine translation. We will discuss more about this in Sect. 2.6.

When we adapt BTG to phrasal translation, we only need the following three rules:

$$\begin{aligned} X &\rightarrow [X_1, X_2] \\ X &\rightarrow \langle X_1, X_2 \rangle \\ X &\rightarrow e/f \end{aligned} \tag{2.2}$$

In comparison with the normal form of BTG as formulated in the Eq. (2.1), two changes are made in the adapted BTG.

- First, e/f represent a target/source phrase which shares the same definition of "phrase" in phrase-based SMT (Koehn et al. 2003), rather than a word.
- Second, in order to be consistent with phrase-based SMT in which null translation is not used (Lopez 2008), we remove the last two rules that involve null translation ϵ in the Eq. (2.1).

We call the SMT formalism built on this adapted BTG as BTG-based SMT.

2.2 A Unified Framework for BTG-Based SMT

We establish a unified framework for BTG-based SMT in this section. We introduce a universal statistical model that estimates scores of the three kinds of BTG rules listed in the Eq. (2.2) with different features. We also describe a CKY-style decoder that does not integrate any language models. Finally, we briefly introduce reordering in BTG-based SMT.

[1] Readers who are interested in more details of SCFG application in machine translation can refer to Chiang (2006) and Lopez (2008).

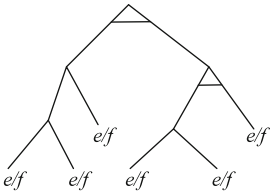

Fig. 2.1 A BTG tree example. The bar under nonterminal nodes indicates that the nodes are generated using an inverted bracketing rule

2.2.1 Model

Given the three BTG rules in the Eq. (2.2), we define a BTG derivation $\mathcal{D}$ as a set of independent applications of lexical and bracketing rules as follows:

$$\mathcal{D} = \langle r^l_{1..n_l}, r^b_{1..n_b} \rangle$$

where $r^l_{1..n_l}$ are lexical rules and $r^b_{1..n_b}$ are bracketing rules. Generally, a BTG derivation can be visualized as a binary BTG tree. Figure 2.1 shows a BTG tree example, where leaf nodes are generated by lexical rules and nonterminal nodes generated by bracketing rules.

We assign a score to each rule using the log-linear model (see Sect. 1.1 of Chap. 1) with different features and corresponding weights λs, then multiply them to obtain the statistical model $M(D)$. To keep in line with the common understanding of standard phrase-based SMT (Koehn et al. 2003), here we reorganize these features into the translation model (M_T), reordering model (M_R), and target language model (P_L) as follows:

$$M(D) = M_T(r^l_{1..n_l}) \cdot M_R(r^b_{1..n_b})^{\lambda_R} \cdot P_L(e)^{\lambda_L} \cdot \exp(|e|)^{\lambda_w} \tag{2.3}$$

where $\exp(|e|)$ is the word penalty, λ_R, λ_L and λ_w are the weight of the reordering model, language model, and word penalty model, respectively.

The translation model M_T is defined as:

$$\begin{aligned} M_T(r^l_{1..n_l}) &= \prod_{i=1}^{n_l} W(r^l_i) \\ W(r^l) &= P(x|y)^{\lambda_1} \cdot P(y|x)^{\lambda_2} \cdot p_{\text{lex}}(x|y)^{\lambda_3} \cdot p_{\text{lex}}(y|x)^{\lambda_4} \cdot \exp(1)^{\lambda_5} \end{aligned} \tag{2.4}$$

where $W(r)$ is the weight of rule r, $P(\cdot)$ represent the phrase translation probabilities in both directions, $p_{\text{lex}}(\cdot)$ denote the lexical translation probabilities in both directions, and $\exp(1)$ is the phrase penalty. Obviously, the translation model is exactly the same as that in standard phrase-based SMT. In other words, the phrase pairs in the phrase table of phrase-based SMT can be directly used as the lexical rules in BTG-based SMT.

The reordering model M_R is defined on the bracketing rules as follows:

$$M_R(r^b_{1..n_b}) = \prod_{i=1}^{n_b} M_R(r^b_i) \tag{2.5}$$

One of the most important and challenging tasks to build a BTG-based SMT system is to develop an appropriate reordering model $M_R(r^b)$ on the bracketing rule r^b. Section 2.2.3 will be devoted to various reordering models for BTG-based SMT.

2.2.2 The −LM Decoding Algorithm

Given an input sentence $f_1 \ldots f_J$, the decoder employs BTG rules (see the Eq. (2.2)) to generate derivations for each segment spanning from f_i to f_j. Our goal is to find the best derivation $\mathcal{D}^*$ that covers the whole input sentence. The final translation e^* is produced from the best derivation as follows:

$$\begin{aligned} \mathcal{D}^* &= \underset{f(\mathcal{D})=f_1..f_J}{\text{argmax}} \; M(\mathcal{D}) \\ e^* &= e(\mathcal{D}^*) \end{aligned} \tag{2.6}$$

where $f(\mathcal{D})$ and $e(\mathcal{D})$ are the source and target yields of $\mathcal{D}$, respectively.

Because the integration of a standard n-gram language model into a CKY-style decoder is not as natural as the integration into a standard phrase-based decoder (Koehn et al. 2003), we separate the language model integration from the decoding algorithm in this section in order to provide a clear and preliminary understanding of the decoding process. Sections 2.3 and 2.4 will discuss more deeply on the integration of an n-gram language model and its variants into the decoder. We call the decoder without language model *−LM decoder*.

Following Chiang (2007), we use the deductive proof system (Shieber et al. 1995; Goodman 1999) to describe the −LM decoder. In a deductive proof system, there are two essential elements: weighted item and inference rule. A weighted item is defined as $\mathcal{I} : w$ where $\mathcal{I}$ represents a chart element in a cell, for instance, $[X, i, j]$ which is a nonterminal X spanning from source word i to j, and w is the weight of the element $\mathcal{I}$. We use inference rules to generate new items. For example,

$$\frac{X \to [X_1, X_2] \;\; [X_1, i, k] : w_1 \;\; [X_2, k+1, j] : w_2}{[X, i, j] : w_1 w_2 P(X \to [X_1, X_2])}$$

The meaning of an inference rule is that if all terms (items or productions) on the top line is true, we can obtain the item on the bottom line. Therefore, the example mentioned above means that if there are an item with weight w_1 spanning from i to k, and the second item with weight w_2 spanning from $k+1$ to i, we can use

$$\frac{X \rightarrow e/f}{[X,i,j]:w} \tag{2.7}$$

$$\frac{X \rightarrow [X_1,X_2] \quad [X_1,i,k]:w_1 \quad [X_2,k+1,j]:w_2}{[X,i,j]:w_1 w_2 (M_R(X \rightarrow [X_1,X_2]))^{\lambda_R}} \tag{2.8}$$

$$\frac{X \rightarrow \langle X_1,X_2 \rangle \quad [X_1,i,k]:w_1 \quad [X_2,k+1,j]:w_2}{[X,i,j]:w_1 w_2 (M_R(X \rightarrow \langle X_1,X_2 \rangle))^{\lambda_R}} \tag{2.9}$$

Fig. 2.2 The −LM decoding algorithm

the production $X \rightarrow [X_1, X_2]$ to combine these two items and generate a new item spanning from i to j with the weight $w_1 w_2 P(X \rightarrow [X_1, X_2])$.

The algorithm of the −LM decoder is shown in Fig. 2.2. The Eq. (2.7) generates an item by using the lexical rule $X \rightarrow e/f$ to translate the source phrase f spanning from i to j to the target phrase e. The weight of the item w is calculated as

$$w = W(X \rightarrow e/f)\text{exp}(|e|)^{\lambda_w}$$

where $W(X \rightarrow e/f)$ is defined in the Eq. (2.4). The Eq. (2.8) combines two neighboring items into a larger item in a straight order and the Eq. (2.9) in an inverted order.

The actual CKY-style decoding procedure is given by the pseudocode in Fig. 2.3. It is easy to prove that the time complexity is $O(J^3)$. We organize all items spanning from i to j into an array $chart[X, i, j]$ (a.k.a. chart cell). First, the chart is initialized

```
1: Procedure CKYDecode
2: for all lexical rules X → e/f do
3:    add item [X,i,j]:w to chart[X,i,j]
4: end for
5: for span = 1 to J do
6:    for i = 1 to J-span+1 do
7:       j = i+span-1
8:       for k = i to j -1 do
9:          for all items [X,i,k]:w1 and [X,k+1,j]:w2 do
10:            add [X,i,j]:w1w2M_R(X → [X1,X2]) to chart[X,i,j]
11:            add [X,i,j]:w1w2M_R(X → ⟨X1,X2⟩) to chart[X,i,j]
12:         end for
13:      end for
14:   end for
15: end for
```

Fig. 2.3 CKY decoding procedure for the −LM decoder

with items generated by applying lexical rules $X \rightarrow e/f$ in which f matches some part of the source sentence. Then for each chart cell that spans from i to j on the source side, all possible derivations over this span are generated. The algorithm guarantees that any subcells within (i, j), such as cell $chart[X, i, k]$ or $chart[X, k+1, j]$, have been filled with items before the chart cell $chart[X, i, j]$ is explored. We generate items for chart cell $chart[X, i, j]$ by using inference rules on items from its any two neighboring subcells $chart[X, i, k]$ and $chart[X, k+1, j]$. We enumerate all possible k so that we explore all combinations of neighboring subcells. These inference rules use the bracketing rules with a *straight* or *inverted* order to generate new items covering span (i, j). The score of a newly generated item is derived from the scores of its two subderivations and the reordering model score according to the Eq. (2.5). When the whole input sentence is covered by items, the decoding is completed.

2.2.3 Reordering

In BTG-based SMT, only two reordering orientations are allowed: either in a straight or an inverted order o when two neighboring nodes X_l and X_r are merged into a larger parent node X_p by a bracketing rule r^b. Therefore, it is natural to define the BTG reordering model $M_R(r^b)$ as a function as follows:[2]

$$M_R(r^b) = h(X_l, X_r, X_p, o) \tag{2.10}$$

where $o \in \{straight, inverted\}$.

Based on this function, various reordering models can be built according to different assumptions. For example, the early used flat reordering model in the original BTG (Wu 1996) assigns prior probabilities for the straight and inverted order assuming the order is highly related to the properties of language pairs. It is formulated as

$$M_R(r^b) = \begin{cases} p_s, & o = straight \\ 1 - p_s, & o = inverted \end{cases} \tag{2.11}$$

Supposing French and English are the source and target language, respectively, the value of p_s can be set as high as 0.8 to prefer monotone orientations since the two languages have similar word orders in most cases.

Similar to the distortion model described by Koehn et al. (2003), we can also define a distortion style reordering model for BTG-based SMT as follows:

$$M_R(r^b) = \begin{cases} \exp(0), & o = straight \\ \exp(\|X_p\|), & o = inverted \end{cases} \tag{2.12}$$

where $\|X_p\|$ denotes the number of words on the source side of node X_p.

[2] X_l, X_r, X_p are actually undifferentiated nonterminals. The subscripts (l, r, p) here are only for notation convenience.

There is a common problem of the flat and distortion reordering model defined above. They do not take any linguistic contexts into account. To be context-dependent, the BTG reordering model might directly estimate the conditional probability as follows:

$$M_R(r^b) = P_R(r^b) = P(o|X_l, X_r, X_p)$$

This probability could be calculated using the maximum likelihood estimate (MLE) by taking counts from training data in the way of lexicalized reordering model (Tillman 2004; Koehn et al. 2005).

$$P(o|X_l, X_r, X_p) = \frac{\text{Count}(o, X_l, X_r, X_p)}{\text{Count}(X_l, X_r, X_p)} \tag{2.13}$$

Unfortunately, this lexicalized reordering method usually suffers from the serious data sparseness problem because X_l, X_r, and X_p become larger and larger as we recursively generate them by combining their children nodes with the bracketing rules, and finally unseen in the training data.

To avoid data sparseness problem, yet be contextually informative, we take a new perspective of reordering in BTG-based SMT. Since our final purpose is to estimate the probability $P_R(r^b)$ of order $o \in \{straight, inverted\}$ for each bracketing operation, we treat reordering in BTG-based SMT as a binary classification problem where the possible order o between the two children nodes is the target class to be predicted. Statistical classifiers therefore can be used for this order prediction task. We use attributes of nodes X_l, X_r and X_p, instead of nodes themselves, as reordering features in the reordering classifier so that the data sparseness problem of the Eq. (2.13) can be avoided. Chapters 3 and 4 will give more details about this classifier-based reordering.

2.3 *n*-Gram Language Model Integration

The standard n-gram language model (Goodman 2001) assigns a probability to a hypothesis e_1^I in the target language as follows:

$$P(e_1^I) = \prod_{i=1}^{I} P(e_i|e_1^{i-1}) \approx \prod_{i=1}^{I} P(e_i|e_{i-n+1}^{i-1}) \tag{2.14}$$

where the approximation is based on the nth order Markov assumption: the prediction of word e_i is dependent on the preceding $n-1$ words $e_{i-n+1} \ldots e_{i-1}$ instead of the whole context history $e_1 \ldots e_{i-1}$.

As we mention in Sect. 2.2.2, the integration of such a language model into the CKY-style decoder is not as trivial as the integration of other models such as the translation model. It is also different from the integration of the n-gram language model into the standard phrase-based decoder (Koehn et al. 2003) in that the

preceding $n-1$ words are not always fully available for words to be predicted when we integrate the n-gram language model into the CKY-style decoder. For example, when we use the lexical rule to translate a source phrase f_i^j into a target phrase e_*^*, the preceding words for the leftmost word of e_*^* are not available as we currently do not know where e_*^* will be placed in the final hypothesis. This section will introduce an algorithm that integrates the n-gram language model into the CKY-style decoder in polynomial time. We call the new decoder integrated with the n-gram language model +LM decoder. We also introduce various pruning methods to speed up the +LM decoder in Sect. 2.5.

Before we introduce the integration algorithm, we define three functions $\mathcal{P}$, $\mathcal{L}$, and $\mathcal{R}$ on a target string e_u^v ($u < v$). The function $\mathcal{P}$ is defined as follows:

$$\begin{aligned}\mathcal{P}(e_u \dots e_v) = & \underbrace{P(e_u) \dots P(e_{u+n-2}|e_u \dots e_{u+n-3})}_{a} \\ & \times \underbrace{\prod_{u+n-1 \le i \le v} P(e_i|e_{i-1} \dots e_{i-n+1})}_{b}\end{aligned} \tag{2.15}$$

The Eq. (2.15) consists of two parts:

- The first part (a) calculates incomplete n-gram language model probabilities for words e_u to e_{u+n-2} which do not have complete $n-1$ preceding words. That means, we calculate the unigram probability for e_u ($P(e_u)$), bigram probability for e_{u+1} ($P(e_{u+1}|e_u)$) and so on until we take $(n-1)$-gram probability for e_{u+n-2} ($P(e_{u+n-2}|e_u \dots e_{u+n-3})$). This resembles the way in which the language model probability in the future cost is computed in the standard phrase-based SMT decoder (Koehn 2012).
- The second part (b) calculates complete n-gram language model probabilities for word e_{u+n-1} to e_v.

This function is different from Chiang's p function in that the latter function p only calculates language model probabilities for complete n-grams. As we mention before, the preceding context for the current word is either yet to be generated or incomplete in terms of n-grams. The $\mathcal{P}$ function enables us to utilize incomplete preceding context to approximately predict words. Once the preceding $n-1$ words are fully available, we can quickly update language model probabilities in an efficient way that will be introduced later in the integration algorithm.

The other two functions $\mathcal{L}$ and $\mathcal{R}$ are defined as follows:

$$\mathcal{L}(e_u \dots e_v) = \begin{cases} e_u \dots e_{u+n-2}, & \text{if } |e_u^v| \ge n \\ e_u \dots e_v, & \text{otherwise} \end{cases} \tag{2.16}$$

$$\mathcal{R}(e_u \dots e_v) = \begin{cases} e_{v-n+2} \dots e_v, & \text{if } |e_u^v| \ge n \\ e_u \dots e_v, & \text{otherwise} \end{cases} \tag{2.17}$$

They return the leftmost and rightmost $n-1$ words from a string, respectively.

The integration algorithm is shown in Fig. 2.4. The item $[X, i, j; l|r]$ indicates a BTG node X spanning from i to j on the source side with the leftmost|rightmost $n-1$ words $l|r$ on the target side. One difference from the $-$LM decoder (see Fig. 2.2) is that we have to record the leftmost|rightmost $n-1$ words for each item in the $+$LM decoder. These $n-1$ terminal symbols in the target language can be considered as the language model state for an item.

In order to highlight how we integrate the language model, we only display the n-gram language model probability for each item, ignoring all other scores that are displayed in the $-$LM decoding algorithm (Fig. 2.2). The Eq. (2.20) in Fig. 2.4 shows how we calculate the language model probability for a BTG lexicon rule which translates a source phrase c into a target phrase e. The Eqs. (2.21) and (2.22) show how we update the language model probabilities for the two bracketing rules which combine two neighboring phrases in a straight and inverted order, respectively. The fundamental theories behind this update are,

$$\mathcal{P}(e_1 e_2) = \mathcal{P}(e_1)\mathcal{P}(e_2)\frac{\mathcal{P}(\mathcal{R}(e_1)\mathcal{L}(e_2))}{\mathcal{P}(\mathcal{R}(e_1))\mathcal{P}(\mathcal{L}(e_2))} \tag{2.18}$$

$$\mathcal{P}(e_2 e_1) = \mathcal{P}(e_2)\mathcal{P}(e_1)\frac{\mathcal{P}(\mathcal{R}(e_2)\mathcal{L}(e_1))}{\mathcal{P}(\mathcal{R}(e_2))\mathcal{P}(\mathcal{L}(e_1))} \tag{2.19}$$

Whenever two strings e_1 and e_2 are concatenated in a straight or inverted order, we can reuse their $\mathcal{P}$ values ($\mathcal{P}(e_1)$ and $\mathcal{P}(e_2)$) in terms of dynamic programming. Only the probabilities of boundary words (e.g., $\mathcal{R}(e_1)\mathcal{L}(e_2)$ in the Eq. (2.18)) need to

$$\frac{X \rightarrow e/f}{[X, i, j; \mathcal{L}(e)|\mathcal{R}(e)] : \mathcal{P}(e)} \tag{2.20}$$

$$\frac{\begin{array}{l} X \rightarrow [X_1, X_2] \\ [X_1, i, k; \mathcal{L}(e_1)|\mathcal{R}(e_1)] : \mathcal{P}(e_1) \\ [X_2, k+1, j; \mathcal{L}(e_2)|\mathcal{R}(e_2)] : \mathcal{P}(e_2) \end{array}}{[X, i, j; \mathcal{L}(e_1 e_2)|\mathcal{R}(e_1 e_2)] : \mathcal{P}(e_1)\mathcal{P}(e_2)\frac{\mathcal{P}(\mathcal{R}(e_1)\mathcal{L}(e_2))}{\mathcal{P}(\mathcal{R}(e_1))\mathcal{P}(\mathcal{L}(e_2))}} \tag{2.21}$$

$$\frac{\begin{array}{l} X \rightarrow \langle X_1, X_2 \rangle \\ [X_1, i, k; \mathcal{L}(e_1)|\mathcal{R}(e_1)] : \mathcal{P}(e_1) \\ [X_2, k+1, j; \mathcal{L}(e_2)|\mathcal{R}(e_2)] : \mathcal{P}(e_2) \end{array}}{[X, i, j; \mathcal{L}(e_2 e_1)|\mathcal{R}(e_2 e_1)] : \mathcal{P}(e_1)\mathcal{P}(e_2)\frac{\mathcal{P}(\mathcal{R}(e_2)\mathcal{L}(e_1))}{\mathcal{P}(\mathcal{R}(e_2))\mathcal{P}(\mathcal{L}(e_1))}} \tag{2.22}$$

Fig. 2.4 The $+$LM decoding algorithm

Table 2.1 Values of $\mathcal{P}$, $\mathcal{L}$, and $\mathcal{R}$ in a trigram example

Function	Value
e_1	$a_1a_2a_3$
e_2	$b_1b_2b_3$
$\mathcal{R}(e_1)$	a_2a_3
$\mathcal{L}(e_2)$	b_1b_2
$\mathcal{P}(\mathcal{R}(e_1))$	$P(a_2)P(a_3\|a_2)$
$\mathcal{P}(\mathcal{L}(e_2))$	$P(b_1)P(b_2\|b_1)$
$\mathcal{P}(e_1)$	$P(a_1)P(a_2\|a_1)P(a_3\|a_1a_2)$
$\mathcal{P}(e_2)$	$P(b_1)P(b_2\|b_1)P(b_3\|b_1b_2)$
$\mathcal{P}(\mathcal{R}(e_1)\mathcal{L}(e_2))$	$P(a_2)P(a_3\|a_2)$
	$P(b_1\|a_2a_3)P(b_2\|a_3b_1)$
$\mathcal{P}(e_1e_2)$	$P(a_1)P(a_2\|a_1)P(a_3\|a_1a_2)$
	$P(b_1\|a_2a_3)P(b_2\|a_3b_1)P(b_3\|b_1b_2)$

be recalculated since they have complete n-grams after the concatenation. Table 2.1 shows values of $\mathcal{P}$, $\mathcal{L}$, and $\mathcal{R}$ in a 3-gram example which helps to verify the Eq. (2.18). These two equations guarantee that the +LM decoding algorithm can correctly compute the language model probability of a sentence stepwise in a dynamic programming framework.[3]

Because in the update parts in the Eqs. (2.18) and (2.19) both the numerator and denominator have up to $2(n-1)$ terminal symbols, the theoretical time complexity of the +LM decoding algorithm is $\mathcal{O}(J^3|T|^{4(n-1)})$ where T is the target language terminal alphabet. This is the same as the time complexity of Chiang's language model integration (Chiang 2007). Practically, this is very slow. Therefore, we have to use various beam search methods for search space pruning.

2.4 Two Extensions to n-Gram Language Model

Language model is one of the most important knowledge sources for statistical machine translation. It is commonly assumed that the quality of language model has great impact on the fluency of target translations. A great variety of methods can be used to improve the quality of language model. The following approaches are widely adopted in the literature of statistical machine translation.

- *Large language model*. More data is better data. Trillions of English words are used to construct a huge language model in a distributed manner (Brants et al. 2007).

[3] The start-of-sentence symbol $\langle s\rangle$ and end-of-sentence symbol $\langle/s\rangle$ can be easily added to update the final language model probability when a translation hypothesis covering the whole source sentence is completed.

- *Language model adaptation*. Language models are adapted to a new domain where an SMT system trained in a different domain is tested (Zhao et al. 2004).
- *Structured language model*. In order to capture long-distance dependencies, syntax-based language models are trained on constituent parse trees or dependency trees (Charniak et al. 2003; Shen et al. 2008; Hassan et al. 2008).

In this section, we focus on techniques that enable standard n-gram language models to capture rich contexts. Our goal is similar to that of structured language models. However, we do not resort to any additional language resources such as parsers. In particular,

1. We build a *backward n-gram language model* that augments a conventional forward n-gram language model with succeeding words.
2. We build a mutual information *trigger language model* which captures long-distance dependencies that go beyond the scope of standard n-gram language models.

The following two Sects. 2.4.1 and 2.4.2 elaborate the backward language model and trigger language model, respectively, with details on modeling, training procedure, and integration algorithm.

2.4.1 Backward Language Model

In this section, we will introduce the backward language model that predicts current words conditioning on their succeeding words, rather than preceding words. Conventional n-gram language models look at the preceding $n-1$ words when calculating the probability of the current word. We henceforth call the previous $n-1$ words plus the current word as *forward n-grams* and a language model built on forward n-grams as forward n-gram language model. Similarly, *backward n-grams* refer to the succeeding $n-1$ words plus the currrent word. We train a backward n-gram language model on backward n-grams and integrate the forward and backward language models together into the decoder. In doing so, we are able to capture both the preceding and succeeding context of a word to be predicted.

2.4.1.1 Model

Given a sequence of words $e_1^I = (e_1 \dots e_I)$, the backward n-gram language model assigns a probability $P_b(e_1^I)$ to e_1^I conditioning on the succeeding context as follows:

$$P_b(e_1^I) = \prod_{i=1}^{I} P(e_i|e_{i+1}^I) \approx \prod_{i=1}^{I} P(e_i|e_{i+1}^{i+n-1}) \tag{2.23}$$

This is different from the forward n-gram language model formulated in the Eq. (2.14) which instead uses the preceding context.

2.4.1.2 Training

For the convenience of training, we invert the order in each sentence in the training data, i.e., from the original order $(e_1 \dots e_I)$ to the reverse order $(e_I \dots e_1)$. In this way, we can use the same toolkit[4] that we use to train a forward n-gram language model to train a backward n-gram language model without any other changes. To be consistent with training, we also need to reverse the order of translation hypotheses when we access the trained backward language model. Note that the Markov context history of the Eq. (2.23) is $e_{i+n-1} \dots e_{i+1}$ instead of $e_{i+1} \dots e_{i+n-1}$ after we invert the order. The words are the same but the order is completely reversed.

2.4.1.3 Decoding

The decoding algorithm with a backward n-gram language model is similar to the algorithm shown in Fig. 2.4, which integrates a forward n-gram language model into the CKY-style decoder. The biggest difference is that all input strings of the backward language model are in a reverse order. Therefore, we need to redefine the three functions $\mathcal{P}$, $\mathcal{L}$, and $\mathcal{R}$ on a target string in a reverse order.

The function $\mathcal{P}$ is reformulated on a reversed string $e_v \dots e_u$ $(u < v)$ as follows:

$$\begin{aligned}\mathcal{P}(e_v \dots e_u) = & \underbrace{P(e_v) \dots P(e_{v-n+2}|e_v \dots e_{v-n+3})}_{a} \\ & \times \underbrace{\prod_{v-n+1 \geq i \geq u} P(e_i|w_{i+n-1} \dots w_{i+1})}_{b}\end{aligned} \tag{2.24}$$

The part (a) calculates incomplete backward n-gram language model probabilities for word e_v to e_{v-n+2}, while the part (b) estimates complete backward n-gram language model probabilities for word e_{v-n+1} to e_u (see Sect. 2.3 for more details about (a) and (b) on the forward language model).

The other two functions $\mathcal{L}$ and $\mathcal{R}$ are redefined as follows:

$$\mathcal{L}(e_v \dots e_u) = \begin{cases} e_v \dots e_{v-n+2}, & \text{if } |e_v^u| \geq n \\ e_v \dots e_u, & \text{otherwise} \end{cases} \tag{2.25}$$

$$\mathcal{R}(e_v \dots e_u) = \begin{cases} e_{u+n-2} \dots e_u, & \text{if } |e_v^u| \geq n \\ e_v \dots e_u, & \text{otherwise} \end{cases} \tag{2.26}$$

[4] For example, the SRI language modeling toolkit (Stolcke 2002).

The $\mathcal{L}$ and $\mathcal{R}$ function return the leftmost and rightmost $n-1$ words from a string in a reverse order, respectively.

The decoding algorithm with the backward language model is shown in Fig. 2.5, where $\overline{e}$ denotes the reversed e. Once again in (2.30) and (2.31), we calculate the backward language model probability for a reversed string in a dynamic programming manner based on the following equations.

$$\mathcal{P}(\overline{e_1 e_2}) = \mathcal{P}(\overline{e_1})\mathcal{P}(\overline{e_2})\frac{\mathcal{P}(\mathcal{R}(\overline{e_2})\mathcal{L}(\overline{e_1}))}{\mathcal{P}(\mathcal{R}(\overline{e_2}))\mathcal{P}(\mathcal{L}(\overline{e_1}))} \tag{2.27}$$

$$\mathcal{P}(\overline{e_2 e_1}) = \mathcal{P}(\overline{e_1})\mathcal{P}(\overline{e_2})\frac{\mathcal{P}(\mathcal{R}(\overline{e_1})\mathcal{L}(\overline{e_2}))}{\mathcal{P}(\mathcal{R}(\overline{e_1}))\mathcal{P}(\mathcal{L}(\overline{e_2}))} \tag{2.28}$$

Figure 2.6 displays a real-world example to show how we update the score of a five-gram backward language model when two items are merged. Let us take the item X_{01} for instance. This item is the combination of the item X_0 (with target string e_1 = "understand") and X_1 (with target string e_2 = "that Africa can no longer") in a straight order. The backward language model costs (i.e., $\log\mathcal{P}(\overline{e_1})$ and $\log\mathcal{P}(\overline{e_2})$) of item X_0 and X_1 are -4.52 and -9.67, respectively. The leftmost four words of $\overline{e_1}$ (i.e., $\mathcal{L}(\overline{e_1})$) comprise only the single word "understand," while the rightmost four words of $\overline{e_2}$ (i.e., $\mathcal{R}(\overline{e_2})$) are "no can Africa that". According to the Eq. (2.27), the backward language model cost of the item X_{01} should be $(-4.52) + (-9.67) + (-14.71) - (-4.52) - (-11.25) = -13.13$. Similarly, we can easily calculate the backward language model score of item X_{012} when we merge item X_{01} and X_2. Note that we

$$\frac{X \rightarrow e/f}{[X, i, j; \mathcal{L}(\overline{e})|\mathcal{R}(\overline{e})] : \mathcal{P}(\overline{e})} \tag{2.29}$$

$$\frac{\begin{array}{l} X \rightarrow [X_1, X_2] \\ [X_1, i, k; \mathcal{L}(\overline{e_1})|\mathcal{R}(\overline{e_1})] : \mathcal{P}(\overline{e_1}) \\ [X_2, k+1, j; \mathcal{L}(\overline{e_2})|\mathcal{R}(\overline{e_2})] : \mathcal{P}(\overline{e_2}) \end{array}}{[X, i, j; \mathcal{L}(\overline{e_1 e_2})|\mathcal{R}(\overline{e_1 e_2})] : \mathcal{P}(\overline{e_1})\mathcal{P}(\overline{e_2})\frac{\mathcal{P}(\mathcal{R}(\overline{e_2})\mathcal{L}(\overline{e_1}))}{\mathcal{P}(\mathcal{L}(\overline{e_1}))\mathcal{P}(\mathcal{R}(\overline{e_2}))}} \tag{2.30}$$

$$\frac{\begin{array}{l} X \rightarrow \langle X_1, X_2 \rangle \\ [X_1, i, k; \mathcal{L}(\overline{e_1})|\mathcal{R}(\overline{e_1})] : \mathcal{P}(\overline{e_1}) \\ [X_2, k+1, j; \mathcal{L}(\overline{e_2})|\mathcal{R}(\overline{e_2})] : \mathcal{P}(\overline{e_2}) \end{array}}{[X, i, j; \mathcal{L}(\overline{e_2 e_1})|\mathcal{R}(\overline{e_2 e_1})] : \mathcal{P}(\overline{e_1})\mathcal{P}(\overline{e_2})\frac{\mathcal{P}(\mathcal{R}(\overline{e_1})\mathcal{L}(\overline{e_2}))}{\mathcal{P}(\mathcal{L}(\overline{e_2}))\mathcal{P}(\mathcal{R}(\overline{e_1}))}} \tag{2.31}$$

Fig. 2.5 The +LM decoding algorithm with the backward n-gram language model

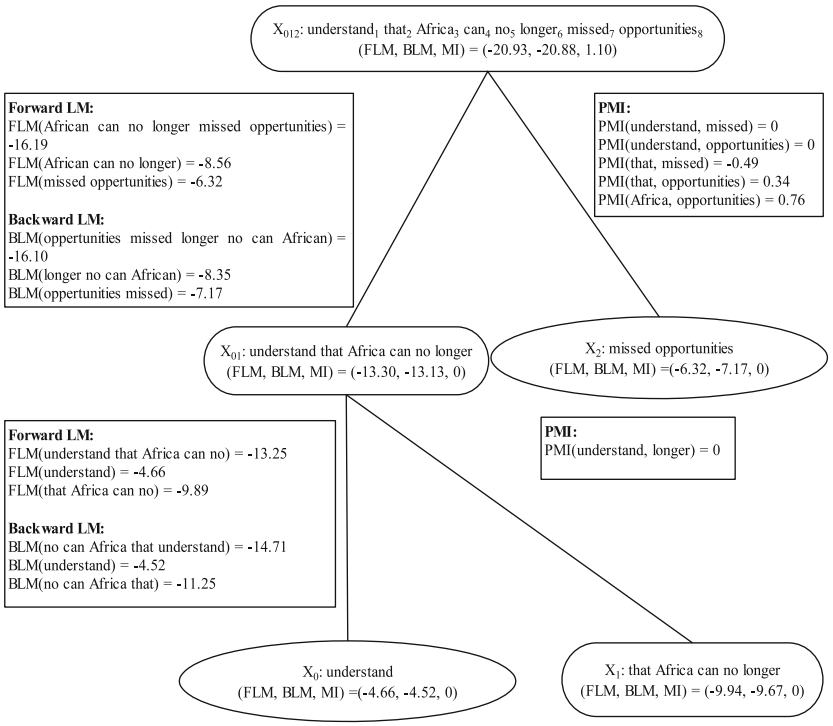

Fig. 2.6 A BTG tree fragment showing how the values of the forward (FLM), backward (BLM), and mutual information trigger (MI) language model are computed. The rectangles display the FLM/BLM costs (log (base-10) probabilities) of target strings as well as the PMI values of trigger pairs that are used to calculate the FLM/BLM/MI values of the item X_{01} (the combination of item X_0 and X_1) and X_{012} (the combination of item X_{01} and X_2)

can store the calculated backward language model scores of the leftmost/rightmost words of each item in practice to save time.

We can also integrate both the forward language model and backward language model into the CKY-style decoder at the same time. The decoding algorithm with the two language models is shown in Fig. 2.7.

2.4.2 Trigger Language Model

It is well-known that long-distance dependencies between words are very important for statistical language modeling. However, conventional *n*-gram language models can only capture short-distance dependencies within an *n*-word window. If the current word is indexed as e_i, the farthest word that a conventional forward *n*-gram includes is e_{i-n+1}. In this section, we introduce a *trigger language model* that is capable of

$$\frac{X \rightarrow e/f}{[X,i,j;\mathcal{L}(\overline{e})|\mathcal{R}(\overline{e})] : \mathcal{P}(e)\mathcal{P}(\overline{e})} \tag{2.32}$$

$$\frac{\begin{array}{l} X \rightarrow [X_1, X_2] \\ [X_1,i,k;\mathcal{L}(e_1)|\mathcal{R}(e_1)] : \mathcal{P}(e_1)\mathcal{P}(\overline{e_1}) \\ [X_2,k+1,j;\mathcal{L}(e_2)|\mathcal{R}(e_2)] : \mathcal{P}(e_2)\mathcal{P}(\overline{e_2}) \end{array}}{\begin{array}{r} [X,i,j;\mathcal{L}(e_1e_2)|\mathcal{R}(e_1e_2)] :\mathcal{P}(e_1)\mathcal{P}(e_2)\dfrac{\mathcal{P}(\mathcal{R}(e_1)\mathcal{L}(e_2))}{\mathcal{P}(\mathcal{R}(e_1))\mathcal{P}(\mathcal{L}(e_2))}\times \\ \mathcal{P}(\overline{e_1})\mathcal{P}(\overline{e_2})\dfrac{\mathcal{P}(\mathcal{R}(\overline{e_2})\mathcal{L}(\overline{e_1}))}{\mathcal{P}(\mathcal{R}(\overline{e_2}))\mathcal{P}(\mathcal{L}(\overline{e_1}))} \end{array}} \tag{2.33}$$

$$\frac{\begin{array}{l} X \rightarrow \langle X_1, X_2 \rangle \\ [X_1,i,k;\mathcal{L}(e_1)|\mathcal{R}(e_1)] : \mathcal{P}(e_1)\mathcal{P}(\overline{e_1}) \\ [X_2,k+1,j;\mathcal{L}(e_2)|\mathcal{R}(e_2)] : \mathcal{P}(e_2)\mathcal{P}(\overline{e_2}) \end{array}}{\begin{array}{r} [X,i,j;\mathcal{L}(e_1e_2)|\mathcal{R}(e_1e_2)] :\mathcal{P}(e_1)\mathcal{P}(e_2)\dfrac{\mathcal{P}(\mathcal{R}(e_2)\mathcal{L}(e_1))}{\mathcal{P}(\mathcal{R}(e_2))\mathcal{P}(\mathcal{L}(e_1))}\times \\ \mathcal{P}(\overline{e_1})\mathcal{P}(\overline{e_2})\dfrac{\mathcal{P}(\mathcal{R}(\overline{e_1})\mathcal{L}(\overline{e_2}))}{\mathcal{P}(\mathcal{R}(\overline{e_1}))\mathcal{P}(\mathcal{L}(\overline{e_2}))} \end{array}} \tag{2.34}$$

Fig. 2.7 The +LM decoding algorithm with the forward and backward n-gram language model

detecting long-distance dependencies that go beyond the scope of traditional forward n-grams, e.g., dependencies between e_i and words from e_1 to e_{i-n}.

2.4.2.1 Model

The trigger language model measures the dependency between two words of a trigger pair. Formally, a trigger pair is defined as an ordered two-tuple (x, y) where word x occurs in the preceding context of word y. It can also be denoted in a more visual manner as $x \rightarrow y$ with x being the trigger and y the triggered word.

We use pointwise mutual information (PMI) (Church and Hanks 1990) to measure the strength of the association between x and y, which is defined as follows:

$$\text{PMI}(x, y) = \log\left(\frac{P(x,y)}{P(x)P(y)}\right) \tag{2.35}$$

Zhou (2004) proposes a new language model that is also enhanced with trigger pairs. In his model, the probability of a given sentence e_1^m is approximated as

$$P(e_1^m) \approx (\prod_{i=1}^{m} P(e_i|e_{i-n+1}^{i-1})) \times \prod_{i=n+1}^{m} \prod_{k=1}^{i-n} \exp(\text{PMI}(e_k, e_i, i-k-1)) \quad (2.36)$$

There are two components in his model. The first component is still the standard n-gram language model. The second one is the mutual information (MI) trigger language model which multiples all exponential PMI values for trigger pairs where the current word is the triggered word and all preceding words outside the n-gram window of the current word are triggers. Note that his MI trigger language model is distance-dependent since trigger pairs (e_k, e_i) are sensitive to their distance $i-k-1$ (zero distance for adjacent words). Therefore, the distance between word x and word y should be taken into account when calculating their PMI.

In order to avoid serious data sparseness, we adopt a *distance-independent* MI trigger language model as follows:

$$\text{MI}(e_1^m) = \prod_{i=n+1}^{m} \prod_{k=1}^{i-n} \exp(\text{PMI}(e_k, e_i)) \quad (2.37)$$

We integrate the distance-independent MI trigger language model into the log-linear model of machine translation as an additional knowledge source which complements the standard n-gram language model in capturing long-distance dependencies. By the minimum error rate training (Och 2003), we are able to tune the weight of the MI trigger langauge model against the weight of the standard n-gram language model while Zhou (2004) sets equal weights for both models.

2.4.2.2 Training

We can use the maximum likelihood estimation method to calculate PMI for each trigger pair by taking counts from training data. Let $C(x, y)$ be the co-occurrence count of the trigger pair (x, y) in the training data. The joint probability of (x, y) is calculated as follows:

$$P(x, y) = \frac{C(x, y)}{\sum_{x,y} C(x, y)} \quad (2.38)$$

The marginal probabilities of x and y can be deduced from the joint probability as follows:

$$P(x) = \sum_{y} P(x, y) \quad (2.39)$$

$$P(y) = \sum_{x} P(x, y) \quad (2.40)$$

Since the number of distinct trigger pairs is $\mathcal{O}(|T|^2)$, the question is how to select valuable trigger pairs. We select trigger pairs according to the following three steps.

(1) The distance between x and y must not be less than $n-1$. Suppose we use a five-gram language model and $y = e_i$, then $x \in \{e_1 \dots e_{i-5}\}$. This is because local dependencies within the n-word window are already captured by the standard n-gram language model. The trigger language model therefore focuses on long-distance dependencies outside the n-word window.
(2) $C(x, y) > c$. We set $c = 10$. This will remove noisy trigger pairs.
(3) Finally, we only keep trigger pairs whose PMI value is larger than 0. Trigger pairs whose PMI value is less than 0 often contain stop words, such as "the", "a". These stop words have very large marginal probabilities due to their high frequencies.

2.4.2.3 Decoding

We integrate the MI trigger model into BTG-based SMT system still in a dynamic programming fashion. In particular, we calculate MI trigger model scores for two strings e_1 and e_2 that are concatenated in a straight ($e_1 e_2$) or inverted ($e_2 e_1$) order as follows:

$$\text{MI}(e_1 e_2) = \text{MI}(e_1)\text{MI}(e_2)\text{MI}(e_1 \mapsto e_2) \tag{2.41}$$

$$\text{MI}(e_2 e_1) = \text{MI}(e_2)\text{MI}(e_1)\text{MI}(e_2 \mapsto e_1) \tag{2.42}$$

where $\text{MI}(e_1 \mapsto e_2)$ represents the PMI values for all trigger pairs in which a word in e_1 triggers a word in e_2. It is defined as follows:

$$\text{MI}(e_1 \mapsto e_2) = \prod_{w_i \in e_2} \prod_{\substack{w_k \in e_1 \\ i-k \geq n}} \exp(\text{PMI}(w_k, w_i)) \tag{2.43}$$

Similarly, we can obtain $\text{MI}(e_2 \mapsto e_1)$ as follows:

$$\text{MI}(e_2 \mapsto e_1) = \prod_{w_i \in e_1} \prod_{\substack{w_k \in e_2 \\ i-k \geq n}} \exp(\text{PMI}(w_k, w_i)) \tag{2.44}$$

The integration algorithm is shown in Fig. 2.8. The problem here is that the state of the MI trigger model involves all words in a string. It is different from the state of n-gram language model as the latter only considers the preceding or succeeding $n-1$ words. If we define the MI trigger model state as the whole string involved, it results in an intractable integration algorithm as we cannot resort to hypotheses recombination for search space pruning. In order to make the integration algorithm tractable, we still use the outermost $n-1$ words to define the MI trigger model state.

$$\frac{X \rightarrow e/f}{[X,i,j] : MI(e)} \tag{2.45}$$

$$\frac{X \rightarrow [X_1, X_2] \quad [X_1,i,k] : MI(e_1) \quad [X_2,k+1,j] : MI(e_2)}{[X,i,j] : MI(e_1)MI(e_2)MI(e_1 \mapsto e_2)} \tag{2.46}$$

$$\frac{X \rightarrow \langle X_1, X_2 \rangle \quad [X_1,i,k] : MI(e_1) \quad [X_2,k+1,j] : MI(e_2)}{[X,i,j] : MI(e_1)MI(e_2)MI(e_2 \mapsto e_1)} \tag{2.47}$$

Fig. 2.8 The decoding algorithm with the trigger language model

Although this will lead to search errors, we can benefit from the MI trigger model for long-distance dependencies in a tractable way.

Let us revisit Fig. 2.6 in order to get a clear picture of how we calculate the mutual information trigger language model. This time, we take the item X_{012} (the combination of item X_{01} and X_2) as an example. The MI trigger language model score of item X_2 (log value of the Eq. (2.37)) is 0 as there are no two words whose distance is equal or larger than 4. For item X_{01}, although the distance of the two words (understand, longer) is equal to 4, they are not selected as a trigger pair as the two words co-occur rarely. Therefore, the default MI value for these two words is set 0. When we combine item X_{01} and X_2 into item X_{012}, the MI trigger language model score of item X_{012} will be deduced from MI(X_{01}), MI(X_2), and all trigger pairs (x, y) where x and y are in X_{01} and X_2, respectively, and their distance is not less than 4. The qualified trigger pairs therefore include (understand, missed), (understand, opportunities), (that, missed), (that, opportunities), and (Africa, opportunities). Their PMI values are shown in the PMI rectangle in Fig. 2.6. As we only use trigger pairs whose PMI value is larger than 0, the final MI trigger language model score of item X_{012} is calculated as MI(X_{01}) + MI(X_2) + PMI(that, opportunities) + PMI(Africa, opportunities) = 1.10 according to the Eq. (2.43).

2.5 Pruning

Search space pruning is very important for SMT decoders. Normally, the following four pruning methods are widely used in SMT systems. We introduce them in the context of BTG-based SMT.

- *Hypothesis recombination*. Whenever two partial hypotheses in the same cell are equivalent, we will recombine them by discarding the one with a lower score. By equivalence, we mean that the two partial hypotheses cover the same span on the source side and contain the same leftmost/rightmost $n - 1$ words on the target

side. Recombination can safely prune hypotheses which will not be included in the final best translation.
- *Threshold pruning*. It discards partial hypotheses with a score worse than α times the best score in the same cell.
- *Histogram pruning*. It only keeps the top N best hypotheses for each cell.
- *Cube pruning*. Suppose we have 100 partial hypotheses exactly covering the source span (i, k) and 100 hypotheses for the source span $(k + 1, j)$. When we use the bracketing rules to combine the two neighboring spans, we will have $2 \times 100 \times 100 = 200{,}000$ new hypotheses which exactly cover the source span (i, j). Most of them will be immediately deleted if we only keep the top 100 best hypotheses for each span. A better way to generate new hypotheses is to only generate hypotheses that have a higher chance to be kept in the top 100. This is the philosophy behind cube pruning that sorts the hypotheses in two neighboring spans by their scores and selects top hypotheses to generate the most promising new hypotheses. This pruning method is first proposed by Chiang (2007).

In this section, we will introduce two variants on the conventional threshold pruning method, both of which speed up the decoding without degrading the performance. The first variant is the dynamic threshold pruning, in which the beam threshold varies with the length of source sequences covered by hypotheses. The second one incorporates a language model dependent probability into the threshold pruning, so that the interaction between a hypothesis and the context outside the hypothesis can be captured.

2.5.1 Dynamic Threshold Pruning

Generally, if we use a loose beam threshold to retain as many hypotheses as possible, the speed of decoding will be very slow, although the translation quality may remain high. On the other hand, if we use a tight beam threshold to prune as many hypotheses as possible, we can get a considerable speedup. However, it comes at a cost of degraded translation quality. Therefore, the question is how we can find an appropriate beam threshold to get the best trade-off between the translation quality and decoding speed. Unfortunately, we are not able to find such an ideal beam threshold since we do not know exactly the distribution of hypotheses beforehand.

Most researchers empirically select a beam threshold on a development set and then use it constantly on a test set. We call this strategy *fixed threshold pruning* (FTP). In order to guarantee a high translation quality, a loose beam threshold is usually used at the cost of slow decoding speed.

A better strategy is to dynamically adjust the beam threshold with a hidden variable. Here, we define the variable as a ratio r (seq/sent) between the length of a source sequence covered by a partial hypothesis and the length of the whole sentence to be translated. To investigate how we should vary the beam threshold with the length

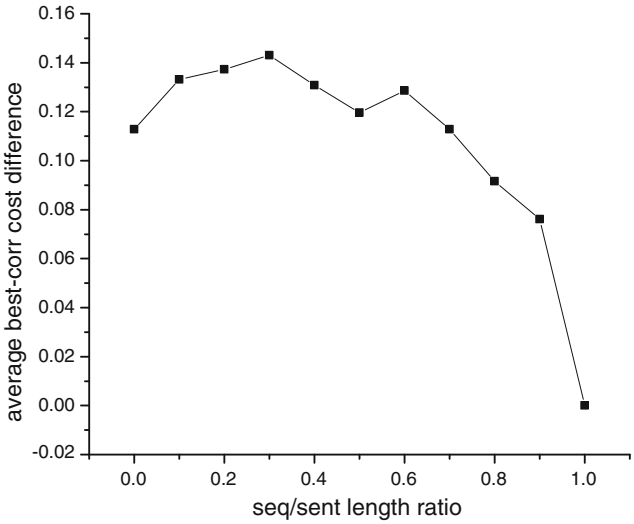

Fig. 2.9 Average best-corr cost difference versus seq/sent length ratio on the NIST MT-02

ratio r, we trace the cost[5] difference (best-corr) between the best hypothesis and the correct hypothesis[6] in chart cells on the NIST MT-02 test set (878 sentences, 19.6 words per sentence). We use a very loose beam threshold[7] to translate sentences on the test set. We plot the curve of average best-corr cost difference versus seq/sent length ratio in Fig. 2.9, which visualizes how wide we should set the beam, so that correct hypotheses fall inside the beam.

From this figure, we can observe that in most cases, the longer the source fragment covered by a hypothesis, the smaller the cost difference between the correct hypothesis and the best hypothesis. This means that we can safely use a tighter beam threshold for hypotheses covering longer source fragments. It is safe because correct hypotheses are still included in the beam, while incorrect hypotheses are pruned as many as possible. However, for hypotheses covering shorter fragments, we should use a looser beam threshold to include all possible candidates for future expansion, so that potential candidates can survive to become part of the finally best hypothesis.

According to this observation, we dynamically adjust the beam threshold parameter α as a function of the length ratio:

$$\alpha = \alpha_0 + (1 - \alpha_0) \cdot r \tag{2.48}$$

[5] The cost of a hypothesis is the negative logarithm of the score of it, estimated by the model shown in the Eq. (2.3). The higher the score, the lower the cost.

[6] The correct hypothesis is the hypothesis that is part of the best translation generated by the decoder. The best hypothesis is the hypothesis with the least cost in the current span. Note that the best hypothesis is not always the correct hypothesis.

[7] Here, we loosen the beam threshold gradually until the BLEU (Papineni et al. 2002) score is not changing. Then, we use the last beam threshold we have tried.

where α_0 is the initial value of the beam threshold parameter which is purposely set to a small value to capture most of the candidates during the early stage of decoding. We call this pruning strategy *dynamic threshold pruning* (DTP). DTP increases the parameter α to tighten the beam when more source words are translated. In theory, DTP runs faster than traditional beam threshold pruning FTP at the same performance level.

2.5.2 LM-Dependent Threshold Pruning

In the traditional beam threshold pruning used in SMT decoding, only the probability estimated from inside a partial hypothesis is adopted. This probability does not provide information about the probability of the hypothesis in the context of the complete translation. In A* decoding for SMT (Och et al. 2001; Zhang and Gildea 2006), different heuristic functions are used to estimate a "future" probability for completing a partial hypothesis. In CKY bottom-up parsing, Goodman (1997) introduces a prior probability into the beam threshold pruning. All of these probabilities are capable of capturing contextual information outside partial hypotheses.

In this section, we introduce an LM-dependent probability for threshold pruning. The basic idea behind the LM-dependent threshold pruning is to incorporate the (forward) language model probability of the boundary words of a hypothesis and neighboring words outside the hypothesis on the target side into the pruning process as early as possible. Since the exact neighboring words are not available until the partial hypothesis is completed, we obtain potential neighboring words in two steps as follows:

- *Step 1*: For each sequence of source words $f_i \ldots f_j$, we find its most probable translation $T(f_i \ldots f_j)$ with a monotone search, only considering the translation model and the language model probability. This can be quickly done with dynamic programming, similar to the method described by Koehn (2004). Then, we cache the leftmost/rightmost target boundary words $T^l(f_i \ldots f_j)/T^r(f_i \ldots f_j)$, which both include $n' = \min(n-1, |T(f_i \ldots f_j)|)$ (n is the language model order) words. Since there are only $J(J+1)/2$ continuous sequences for a source sentence of J words, the target boundary words for all these sequences can be quickly found and cached before decoding with a very cheap overhead.
- *Step 2*: for a hypothesis H covering a source span $f_i \ldots f_j$, we look up the leftmost/rightmost target boundary words of its neighboring spans:

$$T^l(f_1 \ldots f_{i-1})/T^r(f_1 \ldots f_{i-1}) \text{ and } T^l(f_{j+1} \ldots f_J)/T^r(f_{j+1} \ldots s_J)$$

  which are cached in the first step. Although these boundary words are not exactly adjacent to H since there exist thousands of word reorderings, they still provide context information for language model interaction. We utilize them according to the following two reordering options.

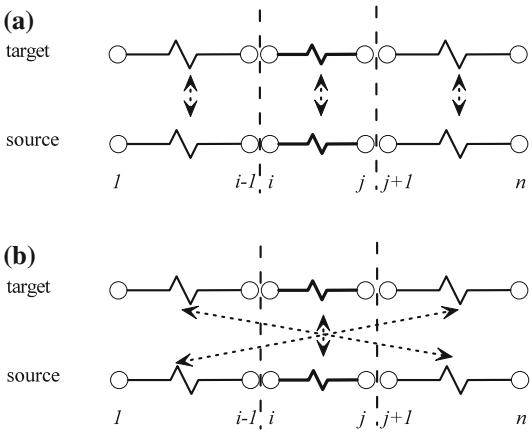

Fig. 2.10 Two reordering options (straight (**a**) and inverted (**b**)) for language model dependent threshold pruning

– If a straight order is preferred (Fig. 2.10a), the LM-dependent threshold probability $\pi_s(H)$ can be estimated as follows:

$$\pi_s(H) = \mathcal{P}(T^r(f_1 \ldots f_{i-1})H^l) \cdot \mathcal{P}(H^r T^l(f_{j+1} \ldots f_J)) \tag{2.49}$$

where $H^{l/r}$ are the leftmost/rightmost boundary words of H, which both include $m' = \min(m-1, |H|)$ words, the function $\mathcal{P}$ is defined in the Eq. (2.15).

– If an inverted order is preferred (Fig. 2.10b), the LM-dependent threshold probability $\pi_i(H)$ can be estimated as follows:

$$\pi_i(H) = \mathcal{P}(T^r(f_{j+1} \ldots f_J)H^l) \cdot \mathcal{P}(H^r T^l(f_1 \ldots f_{i-1})) \tag{2.50}$$

Since we do not know which order will be preferred, we take the maximum of the straight and inverted LM-dependent threshold probability for the hypothesis

$$\pi(H) = \max(\pi_s(H), \pi_i(H)) \tag{2.51}$$

The final beam threshold pruning metric for H when compared to the best hypothesis within the same cell is,

$$M(H) = M_{\text{in}}(H) \cdot \pi(H)^{\lambda_L} \tag{2.52}$$

where $M_{\text{in}}(H)$ is the model score estimated from inside the hypothesis H according to the Eq. (2.3), λ_L is the weight of the language model. Note that $M(H)$ is only used for beam threshold pruning.

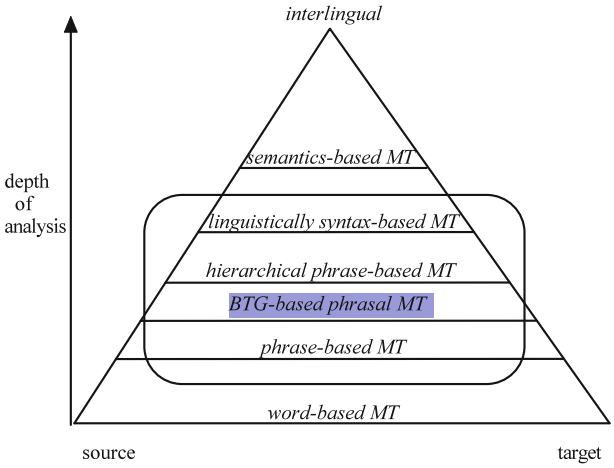

Fig. 2.11 *Triangle* of machine translation

2.6 Comparison with Other SMT Formalisms

In Sects. 2.2 and 2.3, we introduce BTG-based SMT from its inner perspectives: modeling and decoding. We will have a more clear picture of this SMT formalism by comparing it with other SMT formalisms in this section. Figure 2.11 projects BTG-based SMT onto the well-known machine translation triangle. In the figure, we highlight three alternative SMT formalisms: phrase-based SMT, hierarchical phrase-based SMT and linguistically syntax-based SMT. The rest of this section will discuss the differences between BTG-based SMT and the highlighted three alternative formalisms. By this comparison, we will understand more about the strengths and weaknesses of BTG-based SMT.

2.6.1 Comparison with Phrase-Based SMT

The biggest difference between BTG-based SMT and the standard phrase-based SMT (Koehn et al. 2003) is that the former uses two bracketing rules $X \rightarrow [X_1, X_2]$ and $X \rightarrow \langle X_1, X_2 \rangle$ in addition to the lexical rules (i.e., phrase pairs in phrase-based SMT). These two bracketing rules along with the CKY-style decoding endow BTG-based SMT (BTG-SMT) with several advantages over the standard phrase-based SMT (PB-SMT).

2.6.1.1 BTG-SMT Models More Reorderings Than PB-SMT Does

It has been verified that arbitrary reorderings make the decoding search for the best permutation an NP-complete problem, just like the Traveling Salesman Problem as shown by Knight (1999). Therefore, constrained reordering is widely adopted to make the translation decoding computationally tractable. The *ITG constraint* and *IBM constraint* are the two most popular reordering constraints. In the ITG constraint, as shown in Sect. 2.2.3, there are only two reordering options whenever two neighboring phrases are merged: either keeping them in the straight (monotonic) order or inverting the order on the target side. In the IBM constraint, only the k leftmost uncovered words (or phrases in PB-SMT) are allowed to be translated. Typically, k is set to 4.

It is known that the ITG constraint cannot model the so-called "inside-out" reorderings as shown in Fig. 2.12. Although these reorderings do occur in real translations (Wellington et al. 2006), the ITG constraint still allows more phrase reorderings than the IBM constraint if the number of phrases is larger than six according to Zens and Ney (2003). In addition, empirical results described by Zens et al. (2004) further show that the ITG constraint outperforms the IBM constraint in a phrase-based SMT system.

2.6.1.2 BTG-SMT Allows Long-Distance Reordering

As we mention above, the IBM constraint used by PB-SMT typically defines a four-phrase window for local reordering. If we want to allow long-distance reordering, we have to set a larger k or resort to arbitrary permutation, which however makes the decoding more computationally expensive or intractable. In contrast, BTG, as a simple case of SCFG, can easily model long-distance reordering in polynomial time through dynamic programming algorithms (Wu 1996; Lopez 2008).

2.6.1.3 BTG-SMT Facilitates the Incorporation of Syntax into SMT

As linguistic representations of syntax are hierarchical (Chomsky 1957) and BTG-SMT is capable of building hierarchical structures, it is convenient for BTG-SMT to incorporate syntax. However, this is not always true for PB-SMT as the incorporation of syntax into the formalism is mixed with failures (Koehn et al. 2003; Och et al. 2004) and successes (Collins et al. 2005; Wang et al. 2007). It seems that PB-SMT is generally not a good fit for syntax (Lopez 2008).

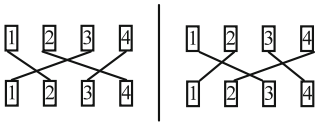

Fig. 2.12 The so-called inside-out reorderings that the ITG constraint does not allow

2.6.2 Comparison with Hierarchical Phrase-Based SMT

We call the grammar induced in hierarchical phrase-based machine translation (Chiang 2005, 2007) *hierarchical grammar*. There are two features of the hierarchical grammar that are also shared by BTG.

1. Only one single undifferentiated nonterminal X is used.
2. At most two nonterminals are permitted at the right-hand side of any productions. Therefore the grammar is an ITG as any SCFG of rank[8] two is an ITG (Wu 2010).

The difference between the hierarchical grammar and BTG is that rules of the hierarchical grammar may contain a number of terminals (words) on the right-hand side for both source and target language as shown in Fig. 2.13. This difference confers upon the hierarchical grammar more modeling power, such as lexicon-sensitive reordering, than BTG. But it comes at a cost of inducing a very large number of rules that are rarely used during decoding (He et al. 2009).

Since both the hierarchical grammar and BTG are an ITG, normally, statistical techniques that are used to improve one of them can be also applied to enhance the other in general.

2.6.3 Comparison with Linguistically Syntax-Based SMT

In recent years, a wide variety of linguistically syntax-based machine translation approaches have been proposed, for example, string-to-tree translation (Galley et al. 2006), tree-to-string translation (Liu et al. 2006), forest-based translation (Mi et al. 2008), dependency-based translation (Quirk et al. 2005; Xiong et al. 2007; Shen et al. 2008) and so on. All these linguistically syntax-based machine translation approaches employ linguistic theories and annotations, which have proven to benefit machine translation.

In contrast to linguistically syntax-based SMT, BTG-based SMT is a formally syntax-based SMT in that it does not depend on any linguistic theories or annotations. On the one hand, such an independence on linguistic theories enables us to train BTG-based SMT more efficiently than linguistically syntax-based SMT because

Fig. 2.13 Sample rules of the hierarchical grammar

$$X \rightarrow \text{grands } X_1/\text{jiyü } X_1$$
$$X \rightarrow \text{held talks } X_1X_2/X_1 \text{ jüxing huitan } X_2$$
$$X \rightarrow \text{accept } X_2 \text{ from } X_1/\text{jieshou } X_1 \text{ de } X_2$$

[8] The *Rank* of an SCFG is the maximum number of nonterminals in the right-hand side of any productions of the synchronous grammar (Chiang 2006).

BTG-based SMT enjoys a much smaller number of synchronous rules. On the other hand, however, BTG-based SMT suffers from no explicit linguistic constraints.

2.6.4 Strengths and Weaknesses of BTG-Based SMT

After comparing BTG-based SMT with the three alternative SMT formalisms (phrase-based SMT, hierarchical phrase-based SMT and linguistically syntax-based SMT), we can summarize the strengths and weaknesses for BTG-based SMT as follows.

- *Strengths*. The formalism has sufficient and flexible reordering ability. It can represent long-distance reordering and build hierarchical structures in polynomial time. It can be trained efficiently.
- *Weakness*. It does not employ any lexical or linguistic syntax knowledge for modeling.

2.7 Summary and Additional Readings

This chapter systematically introduces BTG-based SMT, including the unified framework, reordering under the ITG constraint, the CKY-style −LM decoding algorithm that builds BTG trees from the bottom up, and the +LM decoding algorithm that integrates *n*-gram language models into the decoder for BTG-based SMT. Following this chapter, readers can implement a complete BTG-based SMT system.

This chapter also introduces two extensions to traditional *n*-gram language models, namely the backward language model which uses backward n-grams to predict the current word and the trigger language model which incorporates long-distance trigger pairs into language modeling. The decoding algorithms that integrate the two extended language models into the CKY-style decoder are described. Experiment results of these two extensions are included in "Enhancing Language Models in Statistical Machine Translation with Backward N-grams and Mutual Information Triggers" (Xiong et al. *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, Portland, Oregon, June 19–24, 2011; pp. 1288–1297.).

In order to speed up the decoder, two variants of threshold pruning, i.e., dynamic threshold pruning and LM-dependent threshold pruning, are also introduced. More details about the empirical evaluation of these two pruning methods are given in "Efficient Beam Thresholding for Statistical Machine Translation" (Xiong et al. *Proceedings of the twelfth Machine Translation Summit*, August 26–30, 2009, Ottawa, Ontario, Canada; pp. 363–370.).

Additional Readings. Zhang and Gildea (2005) propose a lexicalized BTG to obtain word alignments. Similarly, Saers et al. (2009) also present an agenda-based BTG biparsing algorithm with a novel pruning method to obtain word alignments. Sánchez

and Benedí (2006) use a stochastic BTG to obtain bilingual phrases for phrase-based SMT. Su et al. (2010) extend the normal BTG to a dependency-based BTG and explore it for machine translation.

Duchateau et al. (2002) use the score estimated by a backward language model in a post-processing step as a confidence measure to detect wrongly recognized words in speech recognition. Finch and Sumita (2009) use a backward language model in their reverse translation decoder where target translations are generated from the ending to the beginning. The backward language model introduced here is different from theirs in that we access the backward language model during decoding (rather than after decoding) where target sentences are still generated from the left to the right.

Rosenfeld et al. (1994) introduce trigger pairs into a maximum entropy based language model as features. The trigger pairs are selected according to their mutual information. Raybaud et al. (2009) use MI triggers in their confidence measures to assess the quality of translation results after decoding. Mauser et al. (2009) propose bilingual triggers where two source words trigger one target word to improve lexical choice of target words.

http://www.springer.com/978-981-287-355-2

Linguistically Motivated Statistical Machine Translation
Models and Algorithms
Xiong, D.; Zhang, M.
2015, XII, 152 p. 52 illus., Hardcover
ISBN: 978-981-287-355-2