Left-to-Right Tree-to-String Decoding with Prediction

Yang Feng[†] **Yang Liu**[‡] **Qun Liu**[⋆] **Trevor Cohn**[†]

[†] Department of Computer Science
The University of Sheffield, Sheffield, UK
{y.feng, t.cohn}@sheffield.ac.uk

[‡] State Key Laboratory on Intelligent Technology and Systems
Tsinghua National Laboratory for Information Science and Technology
Department of Computer Sci. and Tech., Tsinghua University, Beijing, China
liuyang2011@tsinghua.edu.cn

[⋆]Key Laboratory of Intelligent Information Processing
Institute of Computing Technology
Chinese Academy of Sciences, Beijing, China
liuqun@ict.ac.cn

Abstract

Decoding algorithms for syntax based machine translation suffer from high computational complexity, a consequence of intersecting a language model with a context free grammar. Left-to-right decoding, which generates the target string in order, can improve decoding efficiency by simplifying the language model evaluation. This paper presents a novel left to right decoding algorithm for tree-to-string translation, using a bottom-up parsing strategy and dynamic future cost estimation for each partial translation. Our method outperforms previously published tree-to-string decoders, including a competing left-to-right method.

1 Introduction

In recent years there has been rapid progress in the development of tree-to-string models for statistical machine translation. These models use the syntactic parse tree of the source language to inform its translation, which allows the models to capture consistent syntactic transformations between the source and target languages, e.g., from subject-verb-object to subject-object-verb word orderings. Decoding algorithms for grammar-based translation seek to find the best string in the intersection between a weighted context free grammar (the translation mode, given a source string/tree) and a weighted finite state acceptor (an n-gram language model). This intersection is problematic, as it results in an intractably large grammar, and makes exact search impossible.

Most researchers have resorted to approximate search, typically beam search (Chiang, 2007). The decoder parses the source sentence, recording the target translations for each span.[1] As the partial translation hypothesis grows, its component ngrams are scored and the hypothesis score is updated. This decoding method though is inefficient as it requires recording the language model context ($n-1$ words) on the left and right edges of each chart cell. These contexts allow for boundary ngrams to be evaluated when the cell is used in another grammar production. In contrast, if the target string is generated in left-to-right order, then only one language model context is required, and the problem of language model evaluation is vastly simplified.

In this paper, we develop a novel method of left-to-right decoding for tree-to-string translation using a shift-reduce parsing strategy. A central issue in any decoding algorithm is the technique used for pruning the search space. Our left-to-right decoding algorithm groups hypotheses, which cover the same number of source words, into a bin. Pruning requires the evaluation of different hypotheses in the same bin, and elimating the least promising options. As each hypotheses may cover different sets of tree

[1]The process is analogous for tree-to-string models, except that only rules and spans matching those in the source trees are considered. Typically nodes are visited according to a post-order traversal.

Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, pages 1191–1200, Jeju Island, Korea, 12–14 July 2012. ©2012 Association for Computational Linguistics

nodes, it is necessary to consider the cost of uncovered nodes, i.e., the *future cost*. We show that a good future cost estimate is essential for accurate and efficient search, leading to high quality translation output.

Other researchers have also considered the left-to-right decoding algorithm for tree-to-string models. Huang and Mi (2010) developed an Earley-style parsing algorithm (Earley, 1970). In their approach, hypotheses covering the same number of tree nodes were binned together. Their method uses a top-down depth-first search, with a mechanism for early elimation of some rules which lead to dead-ends in the search. Huang and Mi (2010)'s method was shown to outperform the traditional post-order-traversal decoding algorithm, considering fewer hypotheses and thus decoding much faster at the same level of performance. However their algorithm used a very rough estimate of future cost, resulting in more search errors than our approach.

Our experiments show that compared with the Earley-style left-to-right decoding (Huang and Mi, 2010) and the traditional post-order-traversal decoding (Liu et al., 2006) algorithms, our algorithm achieves a significant improvement on search capacity and better translation performance at the same level of speed.

2 Background

A typical tree-to-string system (Liu et al., 2006; Huang et al., 2006) searches through a 1-best source parse tree for the best derivation. It transduces the source tree into a target-language string using a Synchronous Tree Substitution Grammar (STSG). The grammar rules are extracted from bilingual word alignments using the GHKM algorithm (Galley et al., 2004).

We will briefly review the traditional decoding algorithm (Liu et al., 2006) and the Earley-style top-down decoding algorithm (Huang and Mi, 2010) for the tree-to-string model.

2.1 Traditional Decoding

The traditional decoding algorithm processes source tree nodes one by one according to a post-order traversal. For each node, it applies matched STSG rules by substituting each non-terminal with its corresponding translation. For the derivation in Figure 1 (b), the traditional algorithm applies r_2 at node NN_2

$$r_2: \text{NN}_2\ (jieguo) \rightarrow the\ result,$$

to obtain "the result" as the translation of NN_2. Next it applies r_4 at node NP,

$$r_4: \text{NP}\ (\ \text{NN}_1\ (toupiao),\ x_1 : \text{NN}_2\ ) \rightarrow x_1\ of\ the\ vote$$

and replaces NN_2 with its translation "the result", then it gets the translation of NP as "the result of the vote".

This algorithm needs to contain boundary words at both left and right extremities of the target string for the purpose of LM evaluation, which leads to a high time complexity. The time complexity in theory and with beam search (Huang and Mi, 2010) is shown in Table 1.

	in theory	beam search
traditional	$O(nc\|V\|^{4(g-1)})$	$O(ncb^2)$
top-down	$O(c(cr)^d\|V\|^{g-1})$	$O(ncb)$
bottom-up	$O((cr)^d\|V\|^{g-1})$	$O(nub)$

Table 1: Time complexity of different algorithms. *traditional* : Liu et al. (2006), *top-down* : Huang and Mi (2010). n is the source sentence length, b is the beam width, c is the number of rules used for each node, V is the target word vocabulary, g is the order of the language model, d is the depth of the source parse tree, u is the number of viable prefixes for each node and r is the maximum arity of each rule.

2.2 Earley-style Top-down Decoding

The Earley-style decoding algorithm performs a top-down depth-first parsing and generates the target translation left to right. It applies Context-Free Grammar (CFG) rules and employs three actions: *predict*, *scan* and *complete* (Section 3.1 describes how to convert STSG rules into CFG rules). We can simulate its translation process using a stack with a dot . indicating which symbol to process next. For the derivation in Figure 1(b) and CFG rules in Figure 1(c), Figure 2 illustrates the whole translation process.

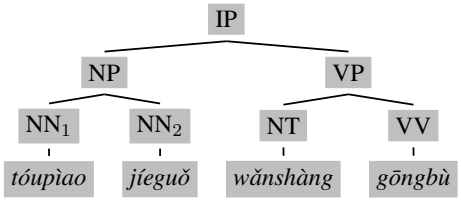
The time complexity is shown in Table 1 .

3 Bottom-Up Left-to-Right Decoding

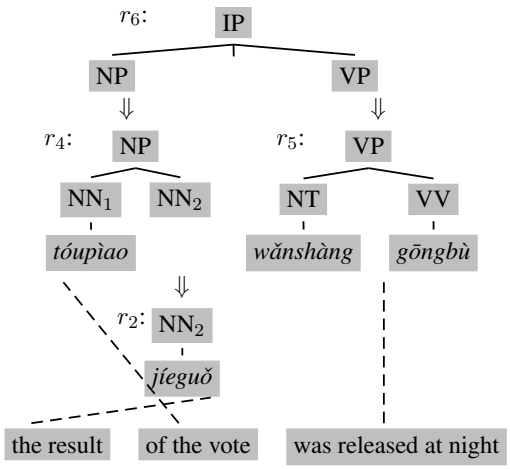
We propose a novel method of left-to-right decoding for tree-to-string translation using a bottom-up parsing strategy. We use **viable prefixes** (Aho and Johnson, 1974) to indicate all possible target strings the translations of each node should starts with. Therefore, given a tree node to expand, our algorithm can drop immediately to target terminals no matter whether there is a **gap** or not. We say that there is a gap between two symbols in a derivation when there are many rules separating them, e.g. IP $\stackrel{r_6}{\rightarrow}$ ... $\stackrel{r_4}{\rightarrow}$ NN_2. For the derivation in Figure 1(b), our algorithm starts from the root node IP and applies r_2 first although there is a gap between IP and NN_2. Then it applies r_4, r_5 and r_6 in sequence to generate the translation "the result of the vote was released at night". Our algorithm takes the gap as a black-box and does not need to fix which partial derivation should be used for the gap at the moment. So it can get target strings as soon as possible and thereby perform more accurate pruning. A valid derivation is generated only when the source tree is completely matched by rules.

Our bottom-up decoding algorithm involves the following steps:

1. Match STSG rules against the source tree.
2. Convert STSG rules to CFG rules.
3. Collect the viable prefix set for each node in a post-order transversal.
4. Search bottom-up for the best derivation.



(a) Source parse tree



(b) A derivation

r_1:	$\text{NN}_1 \rightarrow$ *the vote*
r_2:	$\text{NN}_2 \rightarrow$ *the result*
r_3:	NP $\rightarrow \text{NN}_2$ *of* NN_1
r_4:	NP $\rightarrow \text{NN}_2$ *of the vote*
r_5:	VP $\rightarrow$ *was released at night*
r_6:	IP $\rightarrow$ NP VP
r_7:	IP $\rightarrow \text{NN}_2$ of the vote VP
r_8:	IP $\rightarrow$ VP NP

(c) Target-side CFG rule set

Figure 1: A toy example.

3.1 From STSG to CFG

After rule matching, each tree node has its applicable STSG rule set. Given a matched STSG rule, our decoding algorithm only needs to consider the tree node the rule can be applied to and the target side, so we follow Huang and Mi (2010) to convert STSG rules to CFG rules. For example, an STSG rule

$$\text{NP}\,(\,\text{NN}_1\,(toupiao),\,x_1:\text{NN}_2\,) \rightarrow x_1\;of\;the\;vote$$

can be converted to a CFG rule

$$\text{NP} \rightarrow \text{NN}_2\;of\;the\;vote$$

The target non-terminals are replaced with corresponding source non-terminals. Figure 1 (c) shows all converted CFG rules for the toy example. Note that different STSG rules might be converted to the same CFG rule despite having different source tree structures.

3.2 Viable Prefix

During decoding, how do we decide which rules should be used next given a partial derivation, especially when there is a gap? A key observation is that some rules should be excluded. For example, any derivation for Figure 1(a) will never begin with r_1 as there is no translation starting with "the vote". In order to know which rules can be excluded for each node, we can recursively calculate the starting terminal strings for each node. For example,

NN_1:	{the vote}	NN_2:	{the result}
NT:	$\emptyset$	VV:	$\emptyset$
NP:	{the result}		
VP:	{was released at night}		
IP:	{the result, was released at night}		

Table 2: The Viable prefix sets for Figure 1 (c)

according to r_1, the starting terminal string of the translation for NN_1 is "the vote". According to r_2, the starting terminal string for NN_2 is "the result". According to r_3, the starting terminal string of NP must include that of NN_2. Table 2 lists the starting terminal strings of all nodes in Figure 1(a). As the translations of node IP should begin with either "the result" or "was released at night", the first rule must be either r_2 or r_5. Therefore, r_1 will never be used as the first rule in any derivation.

We refer to starting terminal strings of a node as a **viable prefixes**, a term borrowed from LR parsing (Aho and Johnson, 1974). Viable prefixes are used to decide which rule should be used to ensure efficient left-to-right target generation. Formally, assume that V_N denotes the set of non-terminals (i.e., source tree node labels), V_T denotes the set of terminals (i.e., target words), $v_1, v_2 \in V_N$, $w \in V_T$, $\pi \in \{V_T \cup V_N\}^*$, we say that w is a viable prefix of v_1 if and only if:

- $v_1 \rightarrow w$, or
- $v_1 \rightarrow wv_2\pi$, or
- $v_1 \rightarrow v_2\pi$, and w is a viable prefix of v_2.

Note that we bundle all successive terminals in one symbol.

3.3 Shift-Reduce Parsing

We use a shift-reduce algorithm to search for the best deviation. The algorithm maintains a stack of dotted rules (Earley, 1970). Given the source tree in Figure 1(a), the stack is initialized with a dotted rule for the root node IP:

[. IP].

Then, the algorithm selects one viable prefix of IP and appends it to the stack with the dot at the beginning (*predict*):

[. IP] [. the result][2].

Then, a *scan* action is performed to produce a partial translation "the result":

[. IP] [the result .].

Next, the algorithm searches for the CFG rules starting with "the result" and gets r_2. Then, it pops the rightmost dotted rule and append the left-hand side (LHS) of r_2 to the stack (*complete*):

[. IP] [NN_2 .].

Next, the algorithm chooses r_4 whose right-hand side "NN2 of the vote" matches the rightmost dotted rule in the stack[3] and *grows* the rightmost dotted rule:

[. IP] [NN_2 . of the vote].

Figure 3 shows the whole process of derivation generation.

Formally, we define four actions on the rightmost rule in the stack:

- *Predict*. If the symbol after the dot in the rightmost dotted rule is a non-terminal v, this action chooses a viable prefix w of v and generates a new dotted rule for w with the dot at the beginning. For example:

$$\text{[. IP]} \xrightarrow{predict} \text{[. IP] [. the result]}$$

- *Scan*. If the symbol after the dot in the rightmost dotted rule is a terminal string w, this action advances the dot to update the current partial translation. For example:

$$\text{[. IP] [. the result]} \xrightarrow{scan} \text{[. IP] [the result .]}$$

- *Complete*. If the rightmost dotted rule ends with a dot and it happens to be the right-hand side of a rule, then this action removes the right-most dotted rule. Besides, if the symbol after the dot in the new rightmost rule corresponds to the same tree node as the LHS nonterminal of the rule, this action advance the dot. For example,

$$\text{[. IP] [NP . VP] [was released at night .]} \xrightarrow{complete} \text{[. IP] [NP VP .]}$$

[2]There are another option: "was released at night"

[3]Here there is an alternative: r_3 or r_7

step	action	rule used	stack	hypothesis
0			[. IP]	
1	p	r_6	[. IP] [. NP VP]	
2	p	r_4	[. IP] [. NP VP] [. NN_2 of the vote]	
3	p	r_2	[. IP] [. NP VP] [. NN_2 of the vote] [. the result]	
4	s		[. IP] [. NP VP] [. NN_2 of the vote] [the result .]	the result
5	c		[. IP] [. NP VP] [NN_2 . of the vote]	the result
6	s		[. IP] [. NP VP] [NN_2 of the vote .]	the result of the vote
7	c		[. IP] [NP . VP]	the result of the vote
8	p	r_5	[. IP] [NP . VP] [. was released at night]	the result of the vote
9	s		[. IP] [NP . VP] [was released at night .]	the ... vote was ... night
10	c		[. IP] [NP VP .]	the ... vote was ... night
11	c		[IP .]	the ... vote was ... night

Figure 2: Simulation of top-down translation process for the derivation in Figure 1(b). Actions: p, predict; s, scan; c, complete. "the ... vote" and "was ... released" are the abbreviated form of "the result of the vote" and "was released at night", respectively.

step	action	rule used	stack	number	hypothesis
0			[. IP]	0	
1	p		[. IP] [. the result]	0	
2	s		[. IP] [the result .]	1	the result
3	c	r_2	[. IP] [NN_2 .]	1	the result
4	g	r_4 or r_7	[. IP] [NN_2 . of the vote]	1	the result
5	s		[. IP] [NN_2 of the vote .]	2	the result of the vote
6	c	r_4	[. IP] [NP .]	2	the result of the vote
7	g	r_6	[. IP] [NP . VP]	2	the result of the vote
8	p		[. IP] [NP . VP] [. was released at night]	2	the result of the vote
9	s		[. IP] [NP . VP] [was released at night .]	4	the ... vote was ... night
10	c	r_5	[. IP] [NP VP .]	4	the ... vote was ... night
11	c	r_6	[IP .]	4	the ... vote was ... night

Figure 3: Simulation of bottom-up translation process for the derivation in Figure 1(b). Actions: p, predict; s, scan; c, complete; g, grow. The column of *number* gives the number of source words the hypothesis covers.

If the string cannot rewrite on the frontier non-terminal, then we add the LHS to the stack with the dot after it. For example:

[. IP] [the result .] $\stackrel{complete}{\longrightarrow}$ [. IP] [NN_2 .]

- *Grow*. If the right-most dotted rule ends with a dot and it happens to be the starting part of a CFG rule, this action appends one symbol of the remainder of that rule to the stack [4]. For example:

[. IP] [NN_2 .] $\stackrel{grow}{\longrightarrow}$ [. IP] [NN_2 . of the vote]

From the above definition, we can find that there may be an ambiguity about whether to use a complete action or a grow action. Similarly, predict actions must select a viable prefix form the set for a node. For example in step 5, although we select to perform complete with r_4 in the example, r_7 is applicable, too. In our implementation, if both r_4 and r_7 are applicable, we apply them both to generate two seperate hypotheses. To limit the exponential explosion of hypotheses (Knight, 1999), we use beam search over bins of similar partial hypotheses (Koehn, 2004).

[4] We bundle the successive terminals in one rule into a symbol

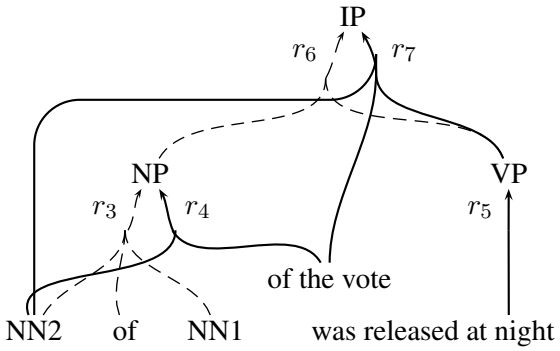


Figure 4: The translation forest composed of applicable CFG rules for the partial derivation of step 3 in Figure 3.

3.4 Future Cost

Partial derivations covering different tree nodes may be grouped in the same bin for beam pruning[5]. In order to perform more accurate pruning, we take into consideration **future cost**, the cost of the uncovered part. The **merit** of a derivation is the **covered cost** (the cost of the covered part) plus the future cost. We borrow ideas from the Inside-Outside algorithm (Charniak and Johnson, 2005; Huang, 2008; Mi et al., 2008) to compute the merit. In our algorithm, the merit of a derivation is just the Viterbi inside cost β of the root node calculated with the derivations continuing from the current derivation.

Given a partial derivation, we calculate its future cost by searching through the translation forest defined by all applicable CFG rules. Figure 4 shows the translation forest for the derivation of step 3. We calculate the future cost for each node as follows: given a node v, we define its cost function $f(v)$ as

$$f(v) = \begin{cases} 1 & v \text{ is completed} \\ lm(v) & v \text{ is a terminal string} \\ \max_{r \in \boldsymbol{R_v}} f(r) \prod_{\pi \in \text{rhs}(r)} f(\pi) & \text{otherwise} \end{cases}$$

where V_N is the non-terminal set, V_T is the terminal set, $v, \pi \in V_N \cup {V_T}^+$, R_v is the set of currently applicable rules for v, $rhs(r)$ is the right-hand symbol set of r, lm is the local language model probability, $f(r)$ is calculated using a linear model whose features are bidirectional translation probabilities and lexical probabilities of r. For the translation forest in Figure 4, if we calculate the future cost of NP with r_4, then

$$\begin{aligned} f(NP) &= f(r_4) \cdot f(NN_2) \cdot lm(of\ the\ vote) \\ &= f(r_4) \cdot 1 \cdot lm(of\ the\ vote) \end{aligned}$$

Note that we calculate lm(of the vote) locally and do not take "the result" derived from NN_2 as the context. The lm probability of "the result" has been included in the covered cost.

As a partial derivation grows, some CFG rules will conflict with the derivation (i.e. inapplicable) and the translation forest will change accordingly. For example, when we reach step 5 from step 3 (see Figure 4 for its translation forest), r_3 is inapplicable and thereby should be ruled out. Then the nodes on the path from the last covered node (it is "of the vote" in step 5) to the root node should update their future cost, as they may employ r_3 to produce the future cost. In step 5, NP and IP should be updated. In this sense, we say that the future cost is dynamic.

3.5 Comparison with Top-Down Decoding

In order to generate the translation "the result" based on the derivation in Figure 1(b), Huang and Mi's top-down algorithm needs to specify which rules to apply starting from the root node until it yields "the result". In this derivation, rule r_6 is applied to IP, r_4 to NP, r_2 to NN_2. That is to say, it needs to represent the partial derivation from IP to NN_2 explicitly. This can be a problem when combined with beam pruning. If the beam size is small, it may discard the intermediate hypotheses and thus never consider the string. In our example with a beam of 1, we must select a rule for IP among r_6, r_7 and r_8 although we do not get any information for NP and VP.

Instead, our bottom-up algorithm allows top-down and bottom-up information to be used together with the help of viable prefixes. This allows us to encode more candidate derivations than the purely top-down method. In the above example, our algorithm does not specify the derivation for the gap from IP and "the result". In fact, all derivations composed of currently applicable rules are allowed. When needed, our algorithm derives the derivation dynamically using applicable rules. So when our algorithm performs pruning at the root node, it has got much more information and consequently introduces fewer pruning errors.

[5]Section 3.7 will describe the binning scheme

3.6 Time Complexity

Assume the depth of the source tree is d, the maximum number of matched rules for each node is c, the maximum arity of each rule is r, the language model order is g and the target-language vocabulary is V, then the time complexity of our algorithm is $O((cr)^d|V|^{g-1})$. Analysis is as follows:

Our algorithm expands partial paths with terminal strings to generate new hypotheses, so the time complexity depends on the number of partial paths used. We split a path which is from the root node to a leaf node with a node on it (called the end node) and get the segment from the root node to the end node as a partial path, so the length of the partial path is not definite with a maximum of d. If the length is $d'(d' \leq d)$, then the number of partial paths is $(cr)^{d'}$. Besides, we use the rightest $g-1$ words to signature each partial path, so we can get $(cr)^{d'}|V|^{g-1}$ states. For each state, the number of viable prefixes produced by predict operation is $c^{d-d'}$, so the total time complexity is $f = O((cr)^{d'}|V|^{g-1}c^{d-d'}) = O(c^d r^{d'}|V|^{g-1}) = O((cr)^d|V|^{g-1})$.

3.7 Beam Search

To make decoding tractable, we employ beam search (Koehn, 2004) and choose "binning" as follows: hypotheses covering the same number of source words are grouped in a bin. When expanding a hypothesis in a beam (bin), we take series of actions until new terminals are appended to the hypothesis, then add the new hypothesis to the corresponding beam. Figure 3 shows the number of source words each hypothesis covers.

Among the actions, only the scan action changes the number of source words each hypothesis covers. Although the complete action does not change source word number, it changes the covered cost of hypotheses. So in our implementation, we take scan and complete as "closure" actions. That is to say, once there are some complete actions after a scan action, we finish all the compete actions until the next action is grow. The predict and grow actions decide which rules can be used to expand hypotheses next, so we update the applicable rule set during these two actions.

Given a source sentence with n words, we maintain n beams, and let each beam hold b hypotheses at most. Besides, we prune viable prefixes of each node up to u, so each hypothesis can expand to u new hypotheses at most, so the time complexity of beam search is $O(nub)$.

4 Related Work

Watanabe et al. (2006) present a novel Earley-style top-down decoding algorithm for hierarchical phrase-based model (Chiang, 2005). Their framework extracts Greibach Normal Form rules only, which always has at least one terminal on the left of each rule, and discards other rules.

Dyer and Resnik (2010) describe a translation model that combines the merits of syntax-based models and phrase-based models. Their decoder works in two passes: for first pass, the decoder collects a context-free forest and performs tree-based source reordering without a LM. For the second pass, the decoder adds a LM and performs bottom-up CKY decoding.

Feng et al. (2010) proposed a shift-reduce algorithm to add BTG constraints to phrase-based models. This algorithm constructs a BTG tree in a reduce-eager manner while the algorithm in this paper searches for a best derivation which must be derived from the source tree.

Galley and Manning (2008) use the shift-reduce algorithm to conduct hierarchical phrase reordering so as to capture long-distance reordering. This algorithm shows good performance on phrase-based models, but can not be applied to syntax-based models directly.

5 Experiments

In the experiments, we use two baseline systems: our in-house tree-to-string decoder implemented according to Liu et al. (2006) (denoted as *traditional*) and the Earley-style top-down decoder implemented according to Huang and Mi (2010) (denoted as *top-down*), respectively. We compare our bottom-up left-to-right decoder (denoted as bottom-up) with the baseline in terms of performance, translation quality and decoding speed with different beam sizes, and search capacity. Lastly, we show the influence of future cost. All systems are implemented in C++.

5.1 Data Setup

We used the FBIS corpus consisting of about 250K Chinese-English sentence pairs as the training set. We aligned the sentence pairs using the GIZA++ toolkit (Och and Ney, 2003) and extracted tree-to-string rules according to the GHKM algorithm (Galley et al., 2004). We used the SRILM toolkit (Stolcke, 2002) to train a 4-gram language model on the Xinhua portion of the GIGAWORD corpus.

We used the 2002 NIST MT Chinese-English test set (571 sentences) as the development set and the 2005 NIST MT Chinese-English test set (1082 sentences) as the test set. We evaluated translation quality using BLEU-metric (Papineni et al., 2002) with case-insensitive n-gram matching up to $n = 4$. We used the standard minimum error rate training (Och, 2003) to tune feature weights to maximize BLEU score on the development set.

5.2 Performance Comparison

Our bottom-up left-to-right decoder employs the same features as the traditional decoder: rule probability, lexical probability, language model probability, rule count and word count. In order to compare them fairly, we used the same beam size which is 20 and employed cube pruning technique (Huang and Chiang, 2005).

We show the results in Table 3. From the results, we can see that the bottom-up decoder outperforms top-down decoder and traditional decoder by 1.1 and 0.8 BLEU points respectively and the improvements are statistically significant using the *sign-test* of Collins et al. (2005) ($p < 0.01$). The improvement may result from dynamically searching for a whole derivation which leads to more accurate estimation of a partial derivation. The additional time consumption of the bottom-up decoder against the top-down decoder comes from dynamic future cost computation.

Next we compare decoding speed versus translation quality using various beam sizes. The results are shown in Figure 5. We can see that our bottom-up decoder can produce better BLEU score at the same decoding speed. At small beams (decoding time around 0.5 second), the improvement of translation quality is much bigger.

System	BLEU(%)	Time (s)
Traditional	29.8	0.84
Top-down	29.5	0.41
Bottom-up	**30.6**	0.81

Table 3: Performance comparison.

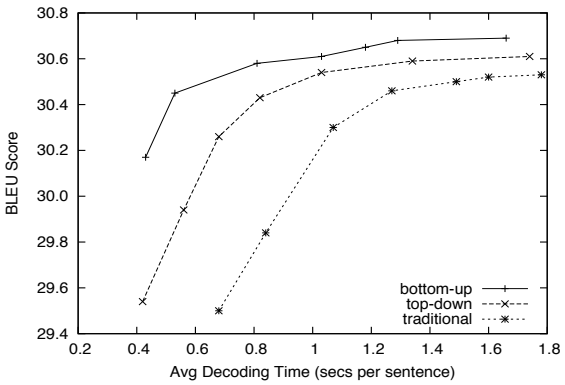


Figure 5: BLEU score against decoding time with various beam size.

5.3 Search Capacity Comparison

We also compare the search capacity of the bottom-up decoder and the traditional decoder. We do this in the following way: we let both decoders use the same weights tuned on the traditional decoder, then we compare their translation scores of the same test sentence.

From the results in Table 4, we can see that for many test sentences, the bottom-up decoder finds target translations with higher score, which have been ruled out by the traditional decoder. This may result from more accurate pruning method. Yet for some sentences, the traditional decoder can attain higher translation score. The reason may be that the traditional decoder can hold more than two nonterminals when cube pruning, while the bottom-up decoder always performs dual-arity pruning.

Next, we check whether higher translation scores bring higher BLEU scores. We compute the BLEU score of both decoders on the test sentence set on which bottom-up decoder gets higher translation scores than the traditional decoder does. We record the results in Figure 6. The result shows that higher score indeed bring higher BLEU score, but the improvement of BLEU score is not large. This is because the features we use don't reflect the real statis-

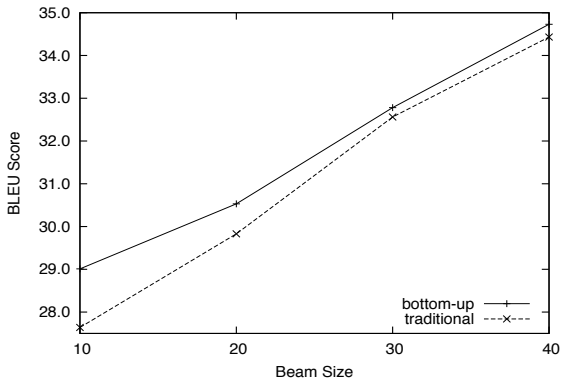


Figure 6: BLEU score with various beam sizes on the sub test set consisting of sentences on which the bottom-up decoder gets higher translation score than the traditional decoder does.

b	$>$		$=$		$<$	
10	728	67%	347	32%	7	1%
20	657	61%	412	38%	13	1%
30	615	57%	446	41%	21	2%
40	526	49%	523	48%	33	3%
50	315	29%	705	65%	62	6%

Table 4: Search capacity comparison. The first column is beam size, the following three columns denote the number of test sentences, on which the translation scores of the bottom-up decoder are greater, equal to, lower than that of the traditional decoder.

System	BLEU(%)	Time (s)
with	30.6	0.81
without	28.8	0.39

Table 5: Influence of future cost. The results of the bottom-up decoder with and without future cost are given in the second and three rows, respectively.

tical distribution of hypotheses well. In addition, the weights are tuned on the traditional decoder, not on the bottom-up decoder. The bottom-up decoder can perform better with weights tuned by itself.

5.4 Influence of Future Cost

Next, we will show the impact of future cost via experiments. We give the results of the bottom-up decoder with and without future cost in Table 5. From the result, we can conclude that future cost plays a significant role in decoding. If the bottom-up decoder does not employ future cost, its performance will be influenced dramatically. Furthermore, calculating dynamic future cost is time consuming. If the bottom-up decoder does not use future cost, it decodes faster than the top-down decoder. This is because the top-down decoder has $|T|$ beams, while the bottom-up decoder has n beams, where T is the source parse tree and n is the length of the source sentence.

6 Conclusions

In this paper, we describe a bottom-up left-to-right decoding algorithm for tree-to-string model. With the help of viable prefixes, the algorithm generates a translation by constructing a target-side CFG tree according to a post-order traversal. In addition, it takes into consideration a dynamic future cost to estimate hypotheses.

On the 2005 NIST Chinese-English MT translation test set, our decoder outperforms the top-down decoder and the traditional decoder by 1.1 and 0.8 BLEU points respectively and shows more powerful search ability. Experiments also prove that future cost is important for more accurate pruning.

7 Acknowledgements

We would like to thank Haitao Mi and Douwe Gelling for their feedback, and anonymous reviewers for their valuable comments and suggestions. This work was supported in part by EPSRC grant EP/I034750/1 and in part by High Technology R&D Program Project No. 2011AA01A207.

References

A. V. Aho and S. C. Johnson. 1974. Lr parsing. *Computing Surveys*, 6:99–124.

Eugene Charniak and Mark Johnson. 2005. Coarse-to-fine n-best parsing and maxent discriminative reranking. In *Proc. of ACL*, pages 173–180.

David Chiang. 2005. A hierarchical phrase-based model for statistical machine translation. In *Proc. of ACL*, pages 263–270.

David Chiang. 2007. Hierarchical phrase-based translation. *Computational Linguistics*, 33:201–228.

Michael Collins, Philipp Koehn, and Ivona Kucerova. 2005. Clause restructuring for statistical machine translation. In *Proc. of ACL*, pages 531–540.

Chris Dyer and Philip Resnik. 2010. Context-free reordering, finite-state translation. In *Proc. of NAACL*, pages 858–866, June.

Jay Earley. 1970. An efficient context-free parsing algorithm. *Communications of the ACM*, 13:94–102.

Yang Feng, Haitao Mi, Yang Liu, and Qun Liu. 2010. An efficient shift-reduce decoding algorithm for phrased-based machine translation. In *Proc. of Coling*, pages 285–293.

Michel Galley and Christopher D. Manning. 2008. A simple and effective hierarchical phrase reordering model. In *Proc. of EMNLP*, pages 848–856.

Michel Galley, Mark Hopkins, Kevin Knight, and Daniel Marcu. 2004. What's in a translation rule? In *Proc of NAACL*, pages 273–280.

Liang Huang and David Chiang. 2005. Better k-best parsing. In *Proc. of IWPT*, pages 53–64.

Liang Huang and Haitao Mi. 2010. Efficient incremental decoding for tree-to-string translation. In *Proc. of EMNLP*, pages 273–283.

Liang Huang, Kevin Knight, and Aravind Joshi. 2006. Statistical syntax-directed translation with extended domain of locality. In *Proceedings of AMTA*.

Liang Huang. 2008. Forest reranking: Discriminative parsing with non-local features. In *Proc. of ACL*, pages 586–594.

Kevin Knight. 1999. Decoding complexity in word-replacement translation models. *Computational Linguistics*, 25:607–615.

Philipp Koehn. 2004. Pharaoh: A beam search decoder for phrased-based statistical machine translation. In *Proc. of AMTA*, pages 115–124.

Yang Liu, Qun Liu, and Shouxun Lin. 2006. Tree-to-string alignment template for statistical machine translation. In *Proceedings of COLING-ACL*, pages 609–616, July.

Haitao Mi, Liang Huang, and Qun Liu. 2008. Forest-based translation. In *Proc. of ACL*, pages 192–199.

Frans J. Och and Hermann Ney. 2003. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29:19–51.

Frans J. Och. 2003. Minimum error rate training in statistical machine translation. In *Proc. of ACL*, pages 160–167.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of ACL*, pages 311–318.

Andreas Stolcke. 2002. Srilm-an extensible language modeling toolkit. In *Proc. of ICSLP*.

Taro Watanabe, Hajime Tsukada, and Hideki Isozaki. 2006. Left-to-right target generation for hierarchical phrase-based translation. In *Proc. of COLING*, pages 777–784.