Efficient Incremental Decoding for Tree-to-String Translation

Liang Huang [1]
[1]Information Sciences Institute
University of Southern California
4676 Admiralty Way, Suite 1001
Marina del Rey, CA 90292, USA
{lhuang,haitaomi}@isi.edu

Haitao Mi [2,1]
[2]Key Lab. of Intelligent Information Processing
Institute of Computing Technology
Chinese Academy of Sciences
P.O. Box 2704, Beijing 100190, China
htmi@ict.ac.cn

Abstract

Syntax-based translation models should *in principle* be efficient with polynomially-sized search space, but in practice they are often embarassingly slow, partly due to the cost of language model integration. In this paper we borrow from phrase-based decoding the idea to generate a translation *incrementally* left-to-right, and show that for tree-to-string models, with a clever encoding of derivation history, this method runs in average-case polynomial-time in theory, and linear-time with beam search in practice (whereas phrase-based decoding is exponential-time in theory and quadratic-time in practice). Experiments show that, with comparable translation quality, our tree-to-string system (in Python) can run more than 30 times faster than the phrase-based system Moses (in C++).

	in theory	*in practice*
phrase-based	exponential	quadratic
tree-to-string	polynomial	linear

Table 1: [main result] Time complexity of our incremental tree-to-string decoding compared with phrase-based. In practice means "approximate search with beams."

1 Introduction

Most efforts in statistical machine translation so far are variants of either phrase-based or syntax-based models. From a theoretical point of view, phrase-based models are neither expressive nor efficient: they typically allow arbitrary permutations and resort to language models to decide the best order. In theory, this process can be reduced to the Traveling Salesman Problem and thus requires an exponential-time algorithm (Knight, 1999). In practice, the decoder has to employ beam search to make it tractable (Koehn, 2004). However, even beam search runs in quadratic-time in general (see Sec. 2), unless a small *distortion limit* (say, d=5) further restricts the possible set of reorderings to those local ones by ruling out any long-distance reorderings that have a "jump" longer than d. This has been the standard practice with phrase-based models (Koehn et al., 2007), which fails to capture important long-distance reorderings like SVO-to-SOV.

Syntax-based models, on the other hand, use syntactic information to restrict reorderings to a computationally-tractable and linguistically-motivated subset, for example those generated by synchronous context-free grammars (Wu, 1997; Chiang, 2007). In theory the advantage seems quite obvious: we can now express global reorderings (like SVO-to-VSO) in polynomial-time (as opposed to exponential in phrase-based). But unfortunately, this polynomial complexity is super-linear (being generally cubic-time or worse), which is slow in practice. Furthermore, language model integration becomes more expensive here since the decoder now has to maintain target-language boundary words at *both* ends of a subtranslation (Huang and Chiang, 2007), whereas a phrase-based decoder only needs to do this at one end since the translation is always growing left-to-right. As a result, syntax-based models are often embarassingly slower than their phrase-based counterparts, preventing them from becoming widely useful.

Can we combine the merits of both approaches? While other authors have explored the possibilities of enhancing phrase-based decoding with syntax-aware reordering (Galley and Manning, 2008), we are more interested in the other direction, i.e., can syntax-based models learn from phrase-based decoding, so that they still model global reordering, but in an efficient (preferably linear-time) fashion?

Watanabe et al. (2006) is an early attempt in this direction: they design a phrase-based-style decoder for the hierarchical phrase-based model (Chiang, 2007). However, this algorithm even with the beam search still runs in quadratic-time in practice. Furthermore, their approach requires grammar transformation that converts the original grammar into an equivalent binary-branching Greibach Normal Form, which is not always feasible in practice.

We take a fresh look on this problem and turn our focus to one particular syntax-based paradigm, tree-to-string translation (Liu et al., 2006; Huang et al., 2006), since this is the simplest and fastest among syntax-based approaches. We develop an incremental dynamic programming algorithm and make the following contributions:

- we show that, unlike previous work, our incremental decoding algorithm runs in average-case **polynomial-time** in theory for tree-to-string models, and the beam search version runs in **linear-time** in practice (see Table 1);
- large-scale experiments on a tree-to-string system confirm that, with comparable translation quality, our incremental decoder (in Python) can run more than 30 times faster than the phrase-based system Moses (in C++) (Koehn et al., 2007);
- furthermore, on the same tree-to-string system, incremental decoding is slightly faster than the standard cube pruning method at the same level of translation quality;
- this is also the first linear-time incremental decoder that performs global reordering.

We will first briefly review phrase-based decoding in this section, which inspires our incremental algorithm in the next section.

2 Background: Phrase-based Decoding

We will use the following running example from Chinese to English to explain both phrase-based and syntax-based decoding throughout this paper:

$_0$ *Bùshí* $_1$	*yǔ* $_2$	*Shālóng* $_3$	*jǔxíng* $_4$	*le* $_5$	*huìtán* $_6$
Bush	with	Sharon	hold	-ed	meeting

‘Bush held talks with Sharon’

2.1 Basic Dynamic Programming Algorithm

Phrase-based decoders generate partial target-language outputs in left-to-right order in the form of *hypotheses* (Koehn, 2004). Each hypothesis has a *coverage vector* capturing the source-language words translated so far, and can be extended into a longer hypothesis by a phrase-pair translating an uncovered segment. This process can be formalized as a deductive system. For example, the following deduction step grows a hypothesis by the phrase-pair $\langle$*yǔ Shālóng*, with Sharon$\rangle$ covering Chinese span [1-3]:

$$\frac{(\bullet\_\_\bullet\bullet\bullet_6) : (w, \text{“Bush held talks”})}{(\bullet\bullet\bullet_3\bullet\bullet\bullet) : (w', \text{“Bush held talks with Sharon”})} \quad (1)$$

where a $\bullet$ in the coverage vector indicates the source word at this position is “covered” and where w and $w' = w+c+d$ are the weights of the two hypotheses, respectively, with c being the cost of the phrase-pair, and d being the *distortion cost*. To compute d we also need to maintain the ending position of the last phrase (the $_3$ and $_6$ in the coverage vector).

To add a bigram model, we split each $-$LM item above into a series of $+$LM items; each $+$LM item has the form $(v, ^a)$ where a is the last word of the hypothesis. Thus a $+$LM version of (1) might be:

$$\frac{(\bullet\_\_\bullet\bullet\bullet_6, ^{\text{talks}}) : (w, \text{“Bush held talks”})}{(\bullet\bullet\bullet_3\bullet\bullet\bullet, ^{\text{Sharon}}) : (w', \text{“Bush held talks with Sharon”})}$$

where the score of the resulting $+$LM item

$$w' = w + c + d - \log P_{lm}(\text{with} \mid \text{talk})$$

now includes a *combination cost* due to the bigrams formed when applying the phrase-pair. The complexity of this dynamic programming algorithm for g-gram decoding is $O(2^n n^2 |V|^{g-1})$ where n is the sentence length and $|V|$ is the English vocabulary size (Huang and Chiang, 2007).

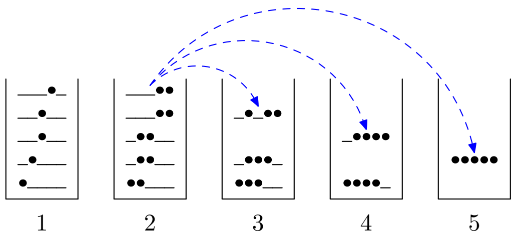


Figure 1: Beam search in phrase-based decoding expands the hypotheses in the current bin (#2) into longer ones.

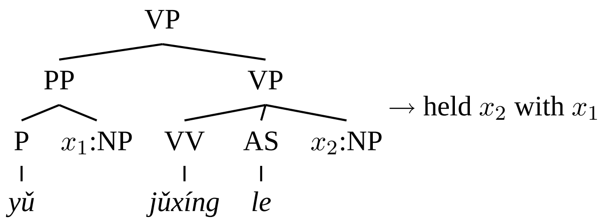


Figure 2: Tree-to-string rule r_3 for reordering.

2.2 Beam Search in Practice

To make the exponential algorithm practical, beam search is the standard approximate search method (Koehn, 2004). Here we group +LM items into n bins, with each bin B_i hosting at most b items that cover exactly i Chinese words (see Figure 1). The complexity becomes $O(n^2b)$ because there are a total of $O(nb)$ items in all bins, and to expand each item we need to scan the whole coverage vector, which costs $O(n)$. This quadratic complexity is still too slow in practice and we often set a small *distortion limit* of $d_{\max}$ (say, 5) so that no jumps longer than $d_{\max}$ are allowed. This method reduces the complexity to $O(nbd_{\max})$ but fails to capture long-distance reorderings (Galley and Manning, 2008).

3 Incremental Decoding for Tree-to-String Translation

We will first briefly review tree-to-string translation paradigm and then develop an incremental decoding algorithm for it inspired by phrase-based decoding.

3.1 Tree-to-string Translation

A typical tree-to-string system (Liu et al., 2006; Huang et al., 2006) performs translation in two steps: parsing and decoding. A parser first parses the source language input into a 1-best tree T, and the decoder then searches for the best *derivation* (a sequence of translation steps) d^* that converts source tree T into a target-language string.

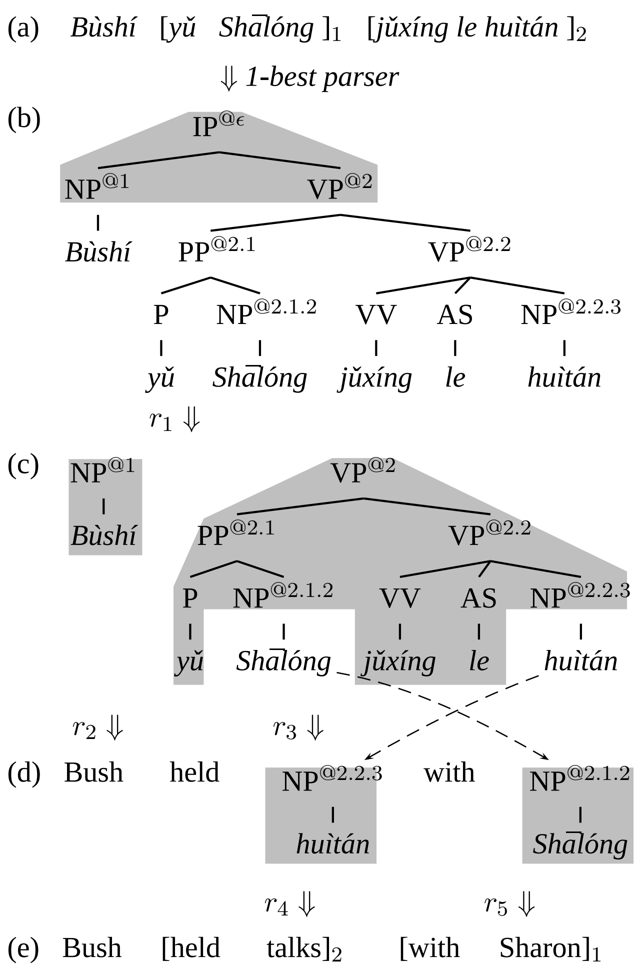


Figure 3: An example derivation of tree-to-string translation (much simplified from Mi et al. (2008)). Shaded regions denote parts of the tree that matches the rule.

Figure 3 shows how this process works. The Chinese sentence (a) is first parsed into tree (b), which will be converted into an English string in 5 steps. First, at the root node, we apply rule r_1 preserving the top-level word-order

(r_1) IP (x_1:NP x_2:VP) → x_1 x_2

which results in two unfinished subtrees, NP$^{@1}$ and VP$^{@2}$ in (c). Here $X^{@\eta}$ denotes a tree node of label X at tree address η (Shieber et al., 1995). (The root node has address ϵ, and the first child of node η has address $\eta.1$, etc.) Then rule r_2 grabs the *Bùshí* subtree and transliterate it into the English word

	in theory	in practice
phrase*	$O(2^n n^2 \cdot \lvert V\rvert^{g-1})$	$O(n^2 b)$
tree-to-str	$O(nc \cdot \lvert V\rvert^{4(g-1)})$	$O(ncb^2)$
*this work**	$O(n^{k \log^2(cr)} \cdot \lvert V\rvert^{g-1})$	$O(ncb)$

Table 2: Summary of time complexities of various algorithms. b is the beam width, V is the English vocabulary, and c is the number of translation rules per node. As a special case, phrase-based decoding with distortion limit $d_{\max}$ is $O(nbd_{\max})$. *: incremental decoding algorithms.

"Bush". Similarly, rule r_3 shown in Figure 2 is applied to the VP subtree, which swaps the two NPs, yielding the situation in (d). Finally two phrasal rules r_4 and r_5 translate the two remaining NPs and finish the translation.

In this framework, decoding without language model ($-$LM decoding) is simply a linear-time depth-first search with memoization (Huang et al., 2006), since a tree of n words is also of size $O(n)$ and we visit every node only once. Adding a language model, however, slows it down significantly because we now have to keep track of target-language boundary words, but unlike the phrase-based case in Section 2, here we have to remember both sides the leftmost and the rightmost boundary words: each node is now split into $+$LM items like $(\eta\ ^{a\ \star\ b})$ where η is a tree node, and a and b are left and right English boundary words. For example, a bigram $+$LM item for node VP$^{@2}$ might be

$$(\text{VP}^{@2\ \text{held}\ \star\ \text{Sharon}}).$$

This is also the case with other syntax-based models like Hiero or GHKM: language model integration overhead is the most significant factor that causes syntax-based decoding to be slow (Chiang, 2007). In theory $+$LM decoding is $O(nc|V|^{4(g-1)})$, where V denotes English vocabulary (Huang, 2007). In practice we have to resort to beam search again: at each node we would only allow top-b $+$LM items. With beam search, tree-to-string decoding with an integrated language model runs in time $O(ncb^2)$, where b is the size of the beam at each node, and c is (maximum) number of translation rules matched at each node (Huang, 2007). See Table 2 for a summary.

3.2 Incremental Decoding

Can we borrow the idea of phrase-based decoding, so that we also grow the hypothesis strictly left-to-right, and only need to maintain the rightmost boundary words?

The key intuition is to adapt the coverage-vector idea from phrase-based decoding to tree-to-string decoding. Basically, a coverage-vector keeps track of which Chinese spans have already been translated and which have not. Similarly, here we might need a "tree coverage-vector" that indicates which subtrees have already been translated and which have not. But unlike in phrase-based decoding, we can not simply choose any arbitrary uncovered subtree for the next step, since rules already dictate which subtree to visit next. In other words what we need here is not really a tree coverage vector, but more of a derivation history.

We develop this intuition into an agenda represented as a stack. Since tree-to-string decoding is a top-down depth-first search, we can simulate this recursion with a stack of active rules, i.e., rules that are not completed yet. For example we can simulate the derivation in Figure 3 as follows. At the root node IP$^{@\epsilon}$, we choose rule r_1, and push its English-side to the stack, with variables replaced by matched tree nodes, here x_1 for NP$^{@1}$ and x_2 for VP$^{@2}$. So we have the following stack

$$s = [\,\centerdot\ \text{NP}^{@1}\ \text{VP}^{@2}],$$

where the dot $\centerdot$ indicates the next symbol to process in the English word-order. Since node NP$^{@1}$ is the first in the English word-order, we expand it first, and push rule r_2 rooted at NP to the stack:

$$[\,\centerdot\ \text{NP}^{@1}\ \text{VP}^{@2}\ ]\ [\,\centerdot\ \text{Bush}].$$

Since the symbol right after the dot in the top rule is a word, we immediately grab it, and append it to the current hypothesis, which results in the new stack

$$[\,\centerdot\ \text{NP}^{@1}\ \text{VP}^{@2}\ ]\ [\text{Bush}\ \centerdot\ ].$$

Now the top rule on the stack has finished (dot is at the end), so we trigger a "pop" operation which pops the top rule and advances the dot in the second-to-top rule, denoting that NP$^{@1}$ is now completed:

$$[\text{NP}^{@1}\ \centerdot\ \text{VP}^{@2}].$$

	stack	hypothesis
	[<s> $\bullet$ IP$^{@\epsilon}$ </s>]	<s>
p	[<s> $\bullet$ IP$^{@\epsilon}$ </s>] [$\bullet$ NP$^{@1}$ VP$^{@2}$]	<s>
p	[<s> $\bullet$ IP$^{@\epsilon}$ </s>] [$\bullet$ NP$^{@1}$ VP$^{@2}$] [$\bullet$ Bush]	<s>
s	[<s> $\bullet$ IP$^{@\epsilon}$ </s>] [$\bullet$ NP$^{@1}$ VP$^{@2}$] [Bush $\bullet$]	<s> Bush
c	[<s> $\bullet$ IP$^{@\epsilon}$ </s>] [NP$^{@1}$ $\bullet$ VP$^{@2}$]	<s> Bush
p	[<s> $\bullet$ IP$^{@\epsilon}$ </s>] [NP$^{@1}$ $\bullet$ VP$^{@2}$] [$\bullet$ held NP$^{@2.2.3}$ with NP$^{@2.1.2}$]	<s> Bush
s	[<s> $\bullet$ IP$^{@\epsilon}$ </s>] [NP$^{@1}$ $\bullet$ VP$^{@2}$] [held $\bullet$ NP$^{@2.2.3}$ with NP$^{@2.1.2}$]	<s> Bush held
p	[<s> $\bullet$ IP$^{@\epsilon}$ </s>] [NP$^{@1}$ $\bullet$ VP$^{@2}$] [held $\bullet$ NP$^{@2.2.3}$ with NP$^{@2.1.2}$] [$\bullet$ talks]	<s> Bush held
s	[<s> $\bullet$ IP$^{@\epsilon}$ </s>] [NP$^{@1}$ $\bullet$ VP$^{@2}$] [held $\bullet$ NP$^{@2.2.3}$ with NP$^{@2.1.2}$] [talks $\bullet$]	<s> Bush held talks
c	[<s> $\bullet$ IP$^{@\epsilon}$ </s>] [NP$^{@1}$ $\bullet$ VP$^{@2}$] [held NP$^{@2.2.3}$ $\bullet$ with NP$^{@2.1.2}$]	<s> Bush held talks
s	[<s> $\bullet$ IP$^{@\epsilon}$ </s>] [NP$^{@1}$ $\bullet$ VP$^{@2}$] [held NP$^{@2.2.3}$ with $\bullet$ NP$^{@2.1.2}$]	<s> Bush held talks with
p	[<s> $\bullet$ IP$^{@\epsilon}$ </s>] [NP$^{@1}$ $\bullet$ VP$^{@2}$] [held NP$^{@2.2.3}$ with $\bullet$ NP$^{@2.1.2}$] [$\bullet$ Sharon]	<s> Bush held talks with
s	[<s> $\bullet$ IP$^{@\epsilon}$ </s>] [NP$^{@1}$ $\bullet$ VP$^{@2}$] [held NP$^{@2.2.3}$ with $\bullet$ NP$^{@2.1.2}$] [Sharon$\bullet$]	<s> Bush held talks with Sharon
c	[<s> $\bullet$ IP$^{@\epsilon}$ </s>] [NP$^{@1}$ $\bullet$ VP$^{@2}$] [held NP$^{@2.2.3}$ with NP$^{@2.1.2}$$\bullet$]	<s> Bush held talks with Sharon
c	[<s> $\bullet$ IP$^{@\epsilon}$ </s>] [NP$^{@1}$ VP$^{@2}$$\bullet$]	<s> Bush held talks with Sharon
c	[<s> IP$^{@\epsilon}$ $\bullet$ </s>]	<s> Bush held talks with Sharon
s	[<s> IP$^{@\epsilon}$ </s>$\bullet$]	<s> Bush held talks with Sharon </s>

Figure 4: Simulation of tree-to-string derivation in Figure 3 in the incremental decoding algorithm. Actions: p, predict; s, scan; c, complete (see Figure 5).

Item $\quad \ell : \langle s,\ \rho\rangle :\ w; \quad \ell$: step, s: stack, ρ: hypothesis, w: weight

Equivalence $\quad \ell : \langle s,\ \rho\rangle \sim \ell : \langle s',\ \rho'\rangle$ iff. $s = s'$ and $last_{g-1}(\rho) = last_{g-1}(\rho')$

Axiom
$$0 : \langle [\texttt{<s>}^{g-1} \bullet \epsilon\ \texttt{</s>}],\ \texttt{<s>}^{g-1}\rangle :\ 0$$

Predict
$$\frac{\ell : \langle ...\ [\alpha \bullet \eta\ \beta],\ \rho\rangle :\ w}{\ell + |C(r)| : \langle ...\ [\alpha \bullet \eta\ \beta]\ [\bullet\ f(\eta, E(r))],\ \rho\rangle :\ w + c(r)} \qquad match(\eta, C(r))$$

Scan
$$\frac{\ell : \langle ...\ [\alpha \bullet e\ \beta],\ \rho\rangle :\ w}{\ell : \langle ...\ [\alpha\ e \bullet \beta],\ \rho e\rangle :\ w - \log \Pr(e \mid last_{g-1}(\rho))}$$

Complete
$$\frac{\ell : \langle ...\ [\alpha \bullet \eta\ \beta]\ [\gamma \bullet],\ \rho\rangle :\ w}{\ell : \langle ...\ [\alpha\ \eta \bullet \beta],\ \rho\rangle :\ w}$$

Goal
$$|T| : \langle [\texttt{<s>}^{g-1}\ \epsilon\ \texttt{</s>}\bullet],\ \rho\texttt{</s>}\rangle :\ w$$

Figure 5: Deductive system for the incremental tree-to-string decoding algorithm. Function $last_{g-1}(\cdot)$ returns the rightmost $g-1$ words (for g-gram LM), and $match(\eta, C(r))$ tests matching of rule r against the subtree rooted at node η. $C(r)$ and $E(r)$ are the Chinese and English sides of rule r, and function $f(\eta, E(r)) = [x_i \mapsto \eta.var(i)]E(r)$ replaces each variable x_i on the English side of the rule with the descendant node $\eta.var(i)$ under η that matches x_i.

The next step is to expand VP$^{@2}$, and we use rule r_3 and push its English-side "VP $\to$ held x_2 with x_1" onto the stack, again with variables replaced by matched nodes:

$$[\text{NP}^{@1} \centerdot \text{VP}^{@2}]\ [\centerdot \text{ held NP}^{@2.2.3} \text{ with NP}^{@2.1.2}]$$

Note that this is a reordering rule, and the stack always follows the English word order because we generate hypothesis incrementally left-to-right. Figure 4 works out the full example.

We formalize this algorithm in Figure 5. Each item $\langle s,\ \rho\rangle$ consists of a stack s and a hypothesis ρ. Similar to phrase-based dynamic programming, only the last $g-1$ words of ρ are part of the signature for decoding with g-gram LM. Each stack is a list of *dotted rules*, i.e., rules with dot positions indicting progress, in the style of Earley (1970). We call the last (rightmost) rule on the stack the *top rule*, which is the rule being processed currently. The symbol after the dot in the top rule is called the *next symbol*, since it is the symbol to expand or process next. Depending on the next symbol a, we can perform one of the three actions:

- if a is a node η, we perform a Predict action which expands η using a rule r that can pattern-match the subtree rooted at η; we push r is to the stack, with the dot at the beginning;
- if a is an English word, we perform a Scan action which immediately adds it to the current hypothesis, advancing the dot by one position;
- if the dot is at the end of the top rule, we perform a Complete action which simply pops stack and advance the dot in the new top rule.

3.3 Polynomial Time Complexity

Unlike phrase-based models, we show here that incremental decoding runs in average-case polynomial-time for tree-to-string systems.

Lemma 1. *For an input sentence of n words and its parse tree of depth d, the worst-case complexity of our algorithm is $f(n,d) = c(cr)^d|V|^{g-1} = O((cr)^d n^{g-1})$, assuming relevant English vocabulary $|V| = O(n)$, and where constants c, r and g are the maximum number of rules matching each tree node, the maximum arity of a rule, and the language-model order, respectively.*

Proof. The time complexity depends (in part) on the number of all possible stacks for a tree of depth d. A stack is a list of rules covering a path from the root node to one of the leaf nodes in the following form:

$$\overbrace{[\ldots \centerdot\eta_1\ldots]}^{R_1}\ \overbrace{[\ldots \centerdot\eta_2\ldots]}^{R_2}\ \ldots\ \overbrace{[\ldots \centerdot\eta_s\ldots]}^{R_s},$$

where $\eta_1 = \epsilon$ is the root node and η_s is a leaf node, with stack depth $s \le d$. Each rule $R_i (i > 1)$ expands node η_{i-1}, and thus has c choices by the definition of grammar constant c. Furthermore, each rule in the stack is actually a dotted-rule, i.e., it is associated with a dot position ranging from 0 to r, where r is the arity of the rule (length of English side of the rule). So the total number of stacks is $O((cr)^d)$.

Besides the stack, each state also maintains $(g-1)$ rightmost words of the hypothesis as the language model signature, which amounts to $O(|V|^{g-1})$. So the total number of states is $O((cr)^d|V|^{g-1})$. Following previous work (Chiang, 2007), we assume a constant number of English translations for each foreign word in the input sentence, so $|V| = O(n)$. And as mentioned above, for each state, there are c possible expansions, so the overall time complexity is $f(n,d) = c(cr)^d|V|^{g-1} = O((cr)^d n^{g-1})$. $\square$

We do average-case analysis below because the tree depth (height) for a sentence of n words is a random variable: in the worst-case it can be linear in n (degenerated into a linear-chain), but we assume this adversarial situation does not happen frequently, and the average tree depth is $O(\log n)$.

Theorem 1. *Assume for each n, the depth of a parse tree of n words, notated d_n, distributes normally with logarithmic mean and variance, i.e., $d_n \sim \mathcal{N}(\mu_n, \sigma_n^2)$, where $\mu_n = O(\log n)$ and $\sigma_n^2 = O(\log n)$, then the average-case complexity of the algorithm is $h(n) = O(n^{k\log^2(cr)+g-1})$ for constant k, thus polynomial in n.*

Proof. From Lemma 1 and the definition of average-case complexity, we have

$$h(n) = \mathbb{E}_{d_n \sim \mathcal{N}(\mu_n, \sigma_n^2)}[f(n, d_n)],$$

where $\mathbb{E}_{x\sim D}[\cdot]$ denotes the expectation with respect to the random variable x in distribution D.

$$\begin{aligned}
h(n) &= \mathbb{E}_{d_n \sim \mathcal{N}(\mu_n, \sigma_n^2)}[f(n, d_n)] \\
&= \mathbb{E}_{d_n \sim \mathcal{N}(\mu_n, \sigma_n^2)}[O((cr)^{d_n} n^{g-1})], \\
&= O(n^{g-1}\mathbb{E}_{d_n \sim \mathcal{N}(\mu_n, \sigma_n^2)}[(cr)^{d_n}]), \\
&= O(n^{g-1}\mathbb{E}_{d_n \sim \mathcal{N}(\mu_n, \sigma_n^2)}[\exp(d_n \log(cr))]) \quad (2)
\end{aligned}$$

Since $d_n \sim \mathcal{N}(\mu_n, \sigma_n^2)$ is a normal distribution, $d_n \log(cr) \sim \mathcal{N}(\mu', \sigma'^2)$ is also a normal distribution, where $\mu' = \mu_n \log(cr)$ and $\sigma' = \sigma_n \log(cr)$. Therefore $\exp(d_n \log(cr))$ is a log-normal distribution, and by the property of log-normal distribution, its expectation is $\exp(\mu' + \sigma'^2/2)$. So we have

$$\begin{aligned}
& \mathbb{E}_{d_n \sim \mathcal{N}(\mu_n, \sigma^2/2)}[\exp(d_n \log(cr))] \\
= & \exp(\mu' + \sigma'^2/2) \\
= & \exp(\mu_n \log(cr) + \sigma_n^2 \log^2(cr)/2) \\
= & \exp(O(\log n)\log(cr) + O(\log n)\log^2(cr)/2) \\
= & \exp(O(\log n)\log^2(cr)) \\
\leq & \exp(k(\log n)\log^2(cr)), \qquad \text{for some constant } k \\
= & \exp(\log n^{k \log^2(cr)}) \\
= & n^{k \log^2(cr)}. \qquad (3)
\end{aligned}$$

Plug it back to Equation (2), and we have the average-case complexity

$$\begin{aligned}
\mathbb{E}_{d_n}[f(n, d_n)] &\leq O(n^{g-1} n^{k \log^2(cr)}) \\
&= O(n^{k \log^2(cr) + g - 1}). \qquad (4)
\end{aligned}$$

Since k, c, r and g are constants, the average-case complexity is polynomial in sentence length n. $\square$

The assumption $d_n \sim \mathcal{N}(O(\log n), O(\log n))$ will be empirically verified in Section 5.

3.4 Linear-time Beam Search

Though polynomial complexity is a desirable property in theory, the degree of the polynomial, $O(\log cr)$ might still be too high in practice, depending on the translation grammar. To make it linear-time, we apply the beam search idea from phrase-based again. And once again, the only question to decide is the choice of "binning": how to assign each item to a particular bin, depending on their progress?

While the number of Chinese words covered is a natural progress indicator for phrase-based, it does not work for tree-to-string because, among the three actions, only scanning grows the hypothesis. The prediction and completion actions do not make real progress in terms of *words*, though they do make progress on the *tree*. So we devise a novel progress indicator natural for tree-to-string translation: the number of tree nodes covered so far. Initially that number is zero, and in a prediction step which expands node η using rule r, the number increments by $|C(r)|$, the size of the Chinese-side treelet of r. For example, a prediction step using rule r_3 in Figure 2 to expand VP$^{@2}$ will increase the tree-node count by $|C(r_3)| = 6$, since there are six tree nodes in that rule (not counting leaf nodes or variables).

Scanning and completion do not make progress in this definition since there is no new tree node covered. In fact, since both of them are deterministic operations, they are treated as "closure" operators in the real implementation, which means that after a prediction, we always do as many scanning/completion steps as possible until the symbol after the dot is another node, where we have to wait for the next prediction step.

This method has $|T| = O(n)$ bins where $|T|$ is the size of the parse tree, and each bin holds b items. Each item can expand to c new items, so the overall complexity of this beam search is $O(ncb)$, which is linear in sentence length.

4 Related Work

The work of Watanabe et al. (2006) is closest in spirit to ours: they also design an incremental decoding algorithm, but for the hierarchical phrase-based system (Chiang, 2007) instead. While we leave detailed comparison and theoretical analysis to a future work, here we point out some obvious differences:

1. due to the difference in the underlying translation models, their algorithm runs in $O(n^2 b)$ time with beam search in practice while ours is linear. This is because each prediction step now has $O(n)$ choices, since they need to expand nodes like VP[1, 6] as:

   $$\text{VP[1,6]} \rightarrow \text{PP}[1, i] \;\; \text{VP}[i, 6],$$

   where the midpoint i in general has $O(n)$ choices (just like in CKY). In other words, their grammar constant c becomes $O(n)$.

2. different binning criteria: we use the number of tree nodes covered, while they stick to the orig-

inal phrase-based idea of number of Chinese words translated;

3. as a result, their framework requires grammar transformation into the binary-branching Greibach Normal Form (which is not always possible) so that the resulting grammar always contain at least one Chinese word in each rule in order for a prediction step to always make progress. Our framework, by contrast, works with any grammar.

Besides, there are some other efforts less closely related to ours. As mentioned in Section 1, while we focus on enhancing syntax-based decoding with phrase-based ideas, other authors have explored the reverse, but also interesting, direction of enhancing phrase-based decoding with syntax-aware reordering. For example Galley and Manning (2008) propose a shift-reduce style method to allow hieararchical non-local reorderings in a phrase-based decoder. While this approach is certainly better than pure phrase-based reordering, it remains quadratic in run-time with beam search.

Within syntax-based paradigms, cube pruning (Chiang, 2007; Huang and Chiang, 2007) has become the standard method to speed up $+$LM decoding, which has been shown by many authors to be highly effective; we will be comparing our incremental decoder with a baseline decoder using cube pruning in Section 5. It is also important to note that cube pruning and incremental decoding are not mutually exclusive, rather, they could potentially be combined to further speed up decoding. We leave this point to future work.

Multipass coarse-to-fine decoding is another popular idea (Venugopal et al., 2007; Zhang and Gildea, 2008; Dyer and Resnik, 2010). In particular, Dyer and Resnik (2010) uses a two-pass approach, where their first-pass, $-$LM decoding is also incremental and polynomial-time (in the style of Earley (1970) algorithm), but their second-pass, $+$LM decoding is still bottom-up CKY with cube pruning.

5 Experiments

To test the merits of our incremental decoder we conduct large-scale experiments on a state-of-the-art tree-to-string system, and compare it with the standard phrase-based system of Moses. Furturemore we also compare our incremental decoder with the standard cube pruning approach on the same tree-to-string decoder.

5.1 Data and System Preparation

Our training corpus consists of 1.5M sentence pairs with about 38M/32M words in Chinese/English, respectively. We first word-align them by GIZA++ and then parse the Chinese sentences using the Berkeley parser (Petrov and Klein, 2007), then apply the GHKM algorithm (Galley et al., 2004) to extract tree-to-string translation rules. We use SRILM Toolkit (Stolcke, 2002) to train a trigram language model with modified Kneser-Ney smoothing on the target side of training corpus. At decoding time, we again parse the input sentences into trees, and convert them into translation forest by rule pattern-matching (Mi et al., 2008).

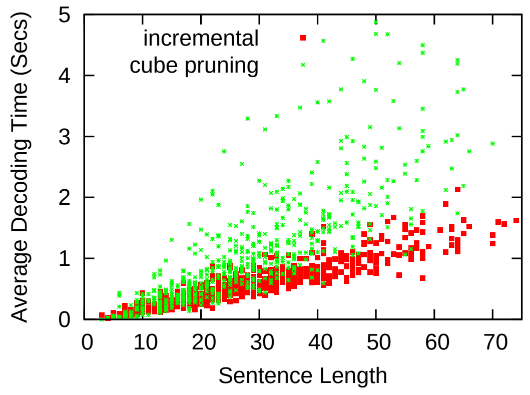
We use the newswire portion of 2006 NIST MT Evaluation test set (616 sentences) as our development set and the newswire portion of 2008 NIST MT Evaluation test set (691 sentences) as our test set. We evaluate the translation quality using the BLEU-4 metric, which is calculated by the script mteval-v13a.pl with its default setting which is case-insensitive matching of n-grams. We use the standard minimum error-rate training (Och, 2003) to tune the feature weights to maximize the system's BLEU score on development set.

We first verify the assumptions we made in Section 3.3 in order to prove the theorem that tree depth (as a random variable) is normally-distributed with $O(\log n)$ mean and variance. Qualitatively, we verified that for most n, tree depth $d(n)$ does look like a normal distribution. Quantitatively, Figure 6 shows that average tree height correlates extremely well with $3.5 \log n$, while tree height variance is bounded by $5.5 \log n$.

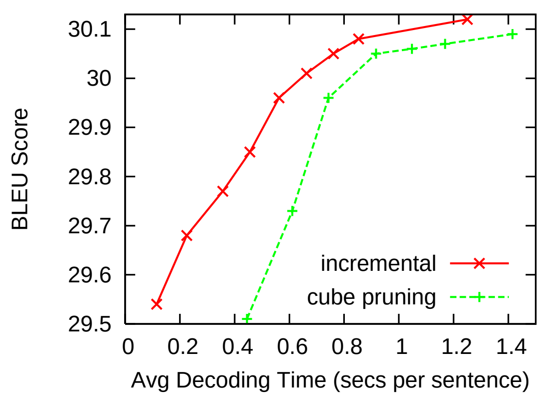
5.2 Comparison with Cube pruning

We implemented our incremental decoding algorithm in Python, and test its performance on the development set. We first compare it with the standard cube pruning approach (also implemented in Python) on the same tree-to-string system.[1] Fig-

[1]Our implementation of cube pruning follows (Chiang, 2007; Huang and Chiang, 2007) where besides a beam size b of unique $+$LM items, there is also a hard limit (of 1000) on the



(a) decoding time against sentence length



(b) BLEU score against decoding time

Figure 7: Comparison with cube pruning. The scatter plot in (a) confirms that our incremental decoding scales linearly with sentence length, while cube pruning super-linearly ($b = 50$ for both). The comparison in (b) shows that at the same level of translation quality, incremental decoding is slightly faster than cube pruning, especially at smaller beams.

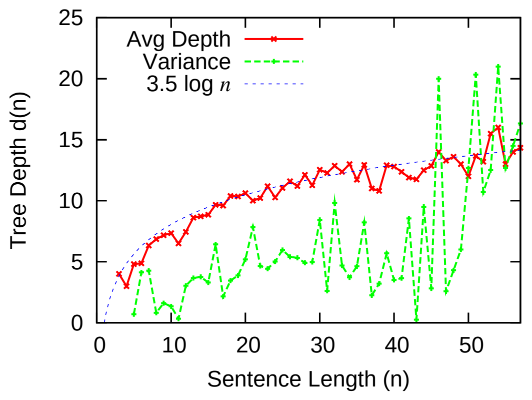


Figure 6: Mean and variance of tree depth vs. sentence length. The mean depth clearly scales with $3.5 \log n$, and the variance is bounded by $5.5 \log n$.

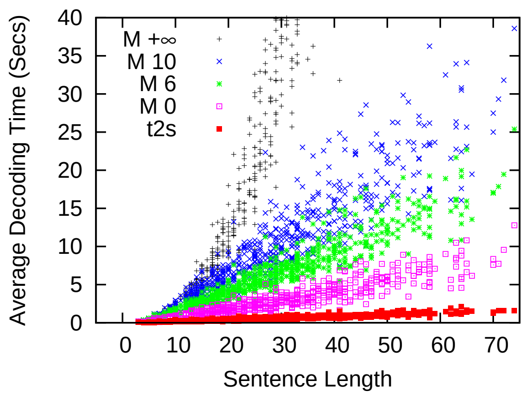


Figure 8: Comparison of our incremental tree-to-string decoder with Moses in terms of speed. Moses is shown with various distortion limits (0, 6, 10, $+\infty$; optimal: 10).

ure 7(a) is a scatter plot of decoding times versus sentence length (using beam $b = 50$ for both systems), where we confirm that our incremental decoder scales linearly, while cube pruning has a slight tendency of superlinearity. Figure 7(b) is a side-by-side comparison of decoding speed versus translation quality (in BLEU scores), using various beam sizes for both systems (b=10–70 for cube pruning, and b=10–110 for incremental). We can see that incremental decoding is slightly faster than cube pruning at the same levels of translation quality, and the difference is more pronounced at smaller beams: for example, at the lowest levels of translation quality (BLEU scores around 29.5), incremental decoding takes only 0.12 seconds, which is about 4 times as fast as cube pruning. We stress again that cube pruning and incremental decoding are not mutually exclusive, and rather they could potentially be combined to further speed up decoding.

number of (non-unique) pops from priority queues.

5.3 Comparison with Moses

We also compare with the standard phrase-based system of Moses (Koehn et al., 2007), with standard settings except for the ttable limit, which we set to 100. Figure 8 compares our incremental decoder

system/decoder	BLEU	time
Moses (optimal $d_{\max}$=10)	29.41	10.8
tree-to-str: cube pruning (b=10)	29.51	0.65
tree-to-str: cube pruning (b=20)	29.96	0.96
tree-to-str: incremental (b=10)	29.54	0.32
tree-to-str: incremental (b=50)	29.96	0.77

Table 3: Final BLEU score and speed results on the test data (691 sentences), compared with Moses and cube pruning. Time is in seconds per sentence, including parsing time (0.21s) for the two tree-to-string decoders.

with Moses at various distortion limits ($d_{\max}$=0, 6, 10, and $+\infty$). Consistent with the theoretical analysis in Section 2, Moses with no distortion limit ($d_{\max} = +\infty$) scale *quadratically*, and monotone decoding ($d_{\max} = 0$) scale linearly. We use MERT to tune the best weights for each distortion limit, and $d_{\max} = 10$ performs the best on our dev set.

Table 3 reports the final results in terms of BLEU score and speed on the test set. Our linear-time incremental decoder with the small beam of size $b = 10$ achieves a BLEU score of 29.54, comparable to Moses with the optimal distortion limit of 10 (BLEU score 29.41). But our decoding (including source-language parsing) only takes 0.32 seconds a sentences, which is more than 30 times faster than Moses. With a larger beam of $b = 50$ our BLEU score increases to 29.96, which is a half BLEU point better than Moses, but still about 15 times faster.

6 Conclusion

We have presented an incremental dynamic programming algorithm for tree-to-string translation which resembles phrase-based based decoding. This algorithm is the first incremental algorithm that runs in polynomial-time in theory, and linear-time in practice with beam search. Large-scale experiments on a state-of-the-art tree-to-string decoder confirmed that, with a comparable (or better) translation quality, it can run more than 30 times faster than the phrase-based system of Moses, even though ours is in Python while Moses in C++. We also showed that it is slightly faster (and scale better) than the popular cube pruning technique. For future work we would like to apply this algorithm to forest-based translation and hierarchical system by pruning the first-pass $-$LM forest. We would also combine cube pruning with our incremental algorithm, and study its performance with higher-order language models.

Acknowledgements

We would like to thank David Chiang, Kevin Knight, and Jonanthan Graehl for discussions and the anonymous reviewers for comments. In particular, we are indebted to the reviewer who pointed out a crucial mistake in Theorem 1 and its proof in the submission. This research was supported in part by DARPA, under contract HR0011-06-C-0022 under subcontract to BBN Technologies, and under DOI-NBC Grant N10AP20031, and in part by the National Natural Science Foundation of China, Contracts 60736014 and 90920004.

References

David Chiang. 2007. Hierarchical phrase-based translation. *Computational Linguistics*, 33(2):201–208.

Chris Dyer and Philip Resnik. 2010. Context-free reordering, finite-state translation. In *Proceedings of NAACL*.

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

Michel Galley and Christopher D. Manning. 2008. A simple and effective hierarchical phrase reordering model. In *Proceedings of EMNLP 2008*.

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

Liang Huang and David Chiang. 2007. Forest rescoring: Fast decoding with integrated language models. In *Proceedings of ACL*, Prague, Czech Rep., June.

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

Liang Huang. 2007. Binarization, synchronous binarization, and target-side binarization. In *Proc. NAACL Workshop on Syntax and Structure in Statistical Translation*.

Kevin Knight. 1999. Decoding complexity in word-replacement translation models. *Computational Linguistics*, 25(4):607–615.

P. Koehn, H. Hoang, A. Birch, C. Callison-Burch, M. Federico, N. Bertoldi, B. Cowan, W. Shen, C. Moran, R. Zens, C. Dyer, O. Bojar, A. Constantin, and E. Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of ACL: demonstration sesion*.

Philipp Koehn. 2004. Pharaoh: a beam search decoder for phrase-based statistical machine translation models. In *Proceedings 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.

Haitao Mi, Liang Huang, and Qun Liu. 2008. Forest-based translation. In *Proceedings of ACL: HLT*, Columbus, OH.

Franz Joseph Och. 2003. Minimum error rate training in statistical machine translation. In *Proceedings of ACL*, pages 160–167.

Slav Petrov and Dan Klein. 2007. Improved inference for unlexicalized parsing. In *Proceedings of HLT-NAACL*.

Stuart Shieber, Yves Schabes, and Fernando Pereira. 1995. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24:3–36.

Andreas Stolcke. 2002. Srilm - an extensible language modeling toolkit. In *Proceedings of ICSLP*, volume 30, pages 901–904.

Ashish Venugopal, Andreas Zollmann, and Stephen Vogel. 2007. An efficient two-pass approach to synchronous-CFG driven statistical MT. In *Proceedings of HLT-NAACL*.

Taro Watanabe, Hajime Tsukuda, and Hideki Isozaki. 2006. Left-to-right target generation for hierarchical phrase-based translation. In *Proceedings of COLING-ACL*.

Dekai Wu. 1997. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational Linguistics*, 23(3):377–404.

Hao Zhang and Daniel Gildea. 2008. Efficient multi-pass decoding for synchronous context free grammars. In *Proceedings of ACL*.