

# Machine Translation as Lexicalized Parsing with Hooks

**Liang Huang**

Dept. of Computer & Information Science  
University of Pennsylvania  
Philadelphia, PA 19104

**Hao Zhang and Daniel Gildea**

Computer Science Department  
University of Rochester  
Rochester, NY 14627

## Abstract

We adapt the “hook” trick for speeding up bilexical parsing to the decoding problem for machine translation models that are based on combining a synchronous context free grammar as the translation model with an  $n$ -gram language model. This dynamic programming technique yields lower complexity algorithms than have previously been described for an important class of translation models.

## 1 Introduction

In a number of recently proposed synchronous grammar formalisms, machine translation of new sentences can be thought of as a form of parsing on the input sentence. The parsing process, however, is complicated by the interaction of the context-free translation model with an  $m$ -gram<sup>1</sup> language model in the output language. While such formalisms admit dynamic programming solutions having polynomial complexity, the degree of the polynomial is prohibitively high.

In this paper we explore parallels between translation and monolingual parsing with lexicalized grammars. Chart items in translation must be augmented with words from the output language in order to capture language model state. This can be thought of as a form of lexicalization with some similarity to that of head-driven lexicalized grammars, despite being unrelated to any notion of syntactic head. We show

<sup>1</sup>We speak of  $m$ -gram language models to avoid confusion with  $n$ , which here is the length of the input sentence for translation.

that techniques for parsing with lexicalized grammars can be adapted to the translation problem, reducing the complexity of decoding with an inversion transduction grammar and a bigram language model from  $O(n^7)$  to  $O(n^6)$ . We present background on this translation model as well as the use of the technique in bilexicalized parsing before describing the new algorithm in detail. We then extend the algorithm to general  $m$ -gram language models, and to general synchronous context-free grammars for translation.

## 2 Machine Translation using Inversion Transduction Grammar

The Inversion Transduction Grammar (ITG) of Wu (1997) is a type of context-free grammar (CFG) for generating two languages synchronously. To model the translational equivalence within a sentence pair, ITG employs a synchronous rewriting mechanism to relate two sentences recursively. To deal with the syntactic divergence between two languages, ITG allows the inversion of rewriting order going from one language to another at any recursive level. ITG in Chomsky normal form consists of unary production rules that are responsible for generating word pairs:

$$X \rightarrow e/f$$

$$X \rightarrow e/\epsilon$$

$$X \rightarrow \epsilon/f$$

where  $e$  is a source language word,  $f$  is a foreign language word, and  $\epsilon$  means the null token, and binary production rules in two forms that are responsible for generating syntactic subtree pairs:

$$X \rightarrow [YZ]$$

and

$$X \rightarrow \langle YZ \rangle$$

The rules with square brackets enclosing the right-hand side expand the left-hand side symbol into the two symbols on the right-hand side in the same order in the two languages, whereas the rules with angled brackets expand the left hand side symbol into the two right-hand side symbols in reverse order in the two languages. The first class of rules is called straight rule. The second class of rules is called inverted rule.

One special case of 2-normal ITG is the so-called Bracketing Transduction Grammar (BTG) which has only one nonterminal  $A$  and two binary rules

$$A \rightarrow [AA]$$

and

$$A \rightarrow \langle AA \rangle$$

By mixing instances of the inverted rule with those of the straight rule hierarchically, BTG can meet the alignment requirements of different language pairs. There exists a more elaborate version of BTG that has 4 nonterminals working together to guarantee the property of one-to-one correspondence between alignments and synchronous parse trees. Table 1 lists the rules of this BTG. In the discussion of this paper, we will consider ITG in 2-normal form.

By associating probabilities or weights with the bitext production rules, ITG becomes suitable for weighted deduction over bitext. Given a sentence pair, searching for the Viterbi synchronous parse tree, of which the alignment is a byproduct, turns out to be a two-dimensional extension of PCFG parsing, having time complexity of  $O(n^6)$ , where  $n$  is the length of the English string and the foreign language string. A more interesting variant of parsing over bitext space is the asymmetrical case in which only the foreign language string is given so that Viterbi parsing involves finding the English string “on the fly”. The process of finding the source string given its target counterpart is decoding. Using ITG, decoding is a form of parsing.

## 2.1 ITG Decoding

Wu (1996) presented a polynomial-time algorithm for decoding ITG combined with an  $m$ -gram lan-

guage model. Such language models are commonly used in noisy channel models of translation, which find the best English translation  $e$  of a foreign sentence  $f$  by finding the sentence  $e$  that maximizes the product of the translation model  $P(f|e)$  and the language model  $P(e)$ .

It is worth noting that since we have specified ITG as a joint model generating both  $e$  and  $f$ , a language model is not theoretically necessary. Given a foreign sentence  $f$ , one can find the best translation  $e^*$ :

$$\begin{aligned} e^* &= \operatorname{argmax}_e P(e, f) \\ &= \operatorname{argmax}_e \sum_q P(e, f, q) \end{aligned}$$

by approximating the sum over parses  $q$  with the probability of the Viterbi parse:

$$e^* = \operatorname{argmax}_e \max_q P(e, f, q)$$

This optimal translation can be computed in using standard CKY parsing over  $f$  by initializing the chart with an item for each possible translation of each foreign word in  $f$ , and then applying ITG rules from the bottom up.

However, ITG’s independence assumptions are too strong to use the ITG probability alone for machine translation. In particular, the context-free assumption that each foreign word’s translation is chosen independently will lead to simply choosing each foreign word’s single most probable English translation with no reordering. In practice it is beneficial to combine the probability given by ITG with a local  $m$ -gram language model for English:

$$e^* = \operatorname{argmax}_e \max_q P(e, f, q) P_{lm}(e)^\alpha$$

with some constant language model weight  $\alpha$ . The language model will lead to more fluent output by influencing both the choice of English words and the reordering, through the choice of straight or inverted rules. While the use of a language model complicates the CKY-based algorithm for finding the best translation, a dynamic programming solution is still possible. We extend the algorithm by storing in each chart item the English *boundary words* that will affect the  $m$ -gram probabilities as the item’s English string is concatenated with the string from an adjacent item. Due to the locality of  $m$ -gram language

	Structural Rules		Lexical Rules
$S \rightarrow A$	$A \rightarrow [AB]$	$B \rightarrow \langle AA \rangle$	$C \rightarrow e_i/f_j$ $C \rightarrow \epsilon/f_j$ $C \rightarrow e_i/\epsilon$
$S \rightarrow B$	$A \rightarrow [BB]$	$B \rightarrow \langle BA \rangle$	
$S \rightarrow C$	$A \rightarrow [CB]$	$B \rightarrow \langle CA \rangle$	
	$A \rightarrow [AC]$	$B \rightarrow \langle AC \rangle$	
	$A \rightarrow [BC]$	$B \rightarrow \langle BC \rangle$	
	$A \rightarrow [CC]$	$B \rightarrow \langle CC \rangle$	

Table 1: Unambiguous BTG

model, only  $m - 1$  boundary words need to be stored to compute the new  $m$ -grams produced by combining two substrings. Figure 1 illustrates the combination of two substrings into a larger one in straight order and inverted order.

### 3 Hook Trick for Bilexical Parsing

A traditional CFG generates words at the bottom of a parse tree and uses nonterminals as abstract representations of substrings to build higher level tree nodes. Nonterminals can be made more specific to the actual substrings they are covering by associating a representative word from the nonterminal’s yield. When the maximum number of lexicalized nonterminals in any rule is two, a CFG is bilexical. A typical bilexical CFG in Chomsky normal form has two types of rule templates:

$$A[h] \rightarrow B[h]C[h']$$

or

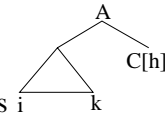
$$A[h] \rightarrow B[h']C[h]$$

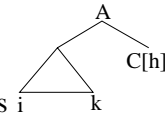
depending on which child is the head child that agrees with the parent on head word selection. Bilexical CFG is at the heart of most modern statistical parsers (Collins, 1997; Charniak, 1997), because the statistics associated with word-specific rules are more informative for disambiguation purposes. If we use  $A[i, j, h]$  to represent a lexicalized constituent,  $\beta(\cdot)$  to represent the Viterbi score function applicable to any constituent, and  $P(\cdot)$  to represent the rule probability function applicable to any rule, Figure 2 shows the equation for the dynamic programming computation of the Viterbi parse. The two terms of the outermost max operator are symmetric cases for heads coming from left and right. Containing five free variables  $i, j, k, h', h$ , ranging over 1 to  $n$ , the length of input sentence, both terms can be

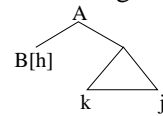
instantiated in  $n^5$  possible ways, implying that the complexity of the parsing algorithm is  $O(n^5)$ .

Eisner and Satta (1999) pointed out we don’t have to enumerate  $k$  and  $h'$  simultaneously. The trick, shown in mathematical form in Figure 2 (bottom) is very simple. When maximizing over  $h'$ ,  $j$  is irrelevant. After getting the intermediate result of maximizing over  $h'$ , we have one less free variable than before. Throughout the two steps, the maximum number of interacting variables is 4, implying that the algorithmic complexity is  $O(n^4)$  after binarizing the factors cleverly. The intermediate result

$$\max_{h', B} [\beta(B[i, k, h']) \cdot P(A[h] \rightarrow B[h']C[h])]$$



can be represented pictorially as . The same trick works for the second max term in Equation 1. The intermediate result coming from binarizing the second term can be visualized as



The shape of the intermediate results gave rise to the nickname of “hook”. Melamed (2003) discussed the applicability of the hook trick for parsing bilexical multitext grammars. The analysis of the hook trick in this section shows that it is essentially an algebraic manipulation. We will formulate the ITG Viterbi decoding algorithm in a dynamic programming equation in the following section and apply the same algebraic manipulation to produce hooks that are suitable for ITG decoding.

### 4 Hook Trick for ITG Decoding

We start from the bigram case, in which each decoding constituent keeps a left boundary word and

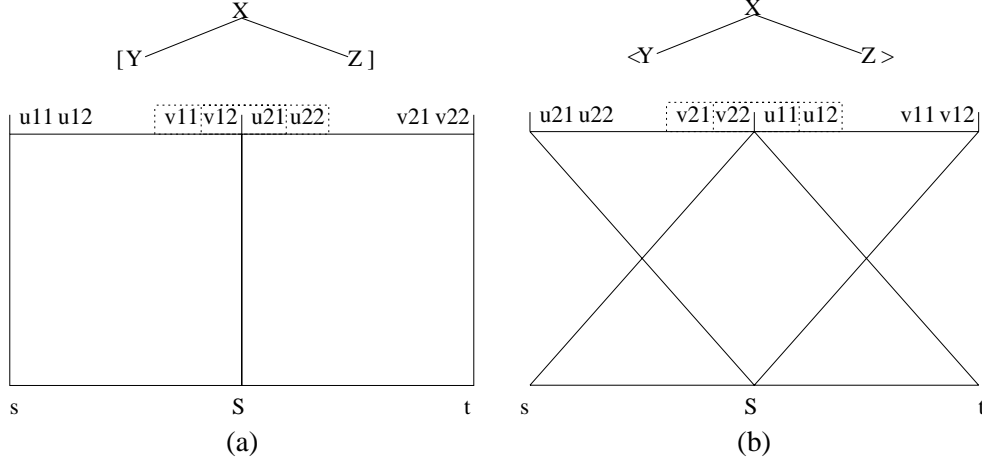


Figure 1: ITG decoding using 3-gram language model. Two boundary words need to be kept on the left ( $u$ ) and right ( $v$ ) of each constituent. In (a), two constituents  $Y$  and  $Z$  spanning substrings  $s, S$  and  $S, t$  of the input are combined using a straight rule  $X \rightarrow [YZ]$ . In (b), two constituents are combined using an inverted rule  $X \rightarrow \langle YZ \rangle$ . The dashed line boxes enclosing three words are the trigrams produced from combining two substrings.

$$\beta(A[i, j, h]) = \max \left\{ \begin{array}{l} \max_{k, h', B, C} \left[ \beta(B[i, k, h']) \cdot \beta(C[k, j, h]) \cdot P(A[h] \rightarrow B[h']C[h]) \right], \\ \max_{k, h', B, C} \left[ \beta(B[i, k, h]) \cdot \beta(C[k, j, h']) \cdot P(A[h] \rightarrow B[h]C[h']) \right] \end{array} \right\} \quad (1)$$

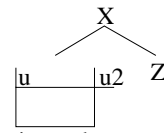
$$\begin{aligned} & \max_{k, h', B, C} \left[ \beta(B[i, k, h']) \cdot \beta(C[k, j, h]) \cdot P(A[h] \rightarrow B[h']C[h]) \right] \\ &= \max_{k, C} \left[ \max_{h', B} \left[ \beta(B[i, k, h']) \cdot P(A[h] \rightarrow B[h']C[h]) \right] \cdot \beta(C[k, j, h]) \right] \end{aligned}$$

Figure 2: Equation for bilexical parsing (top), with an efficient factorization (bottom)

a right boundary word. The dynamic programming equation is shown in Figure 3 (top) where  $i, j, k$  range over 1 to  $n$ , the length of input foreign sentence, and  $u, v, v_1, u_2$  (or  $u, v, v_2, u_1$ ) range over 1 to  $V$ , the size of English vocabulary. Usually we will constrain the vocabulary to be a subset of words that are probable translations of the foreign words in the input sentence. So  $V$  is proportional to  $n$ . There are seven free variables related to input size for doing the maximization computation. Hence the algorithmic complexity is  $O(n^7)$ .

The two terms in Figure 3 (top) within the first level of the max operator, corresponding to straight rules and inverted rules, are analogous to the two terms in Equation 1. Figure 3 (bottom) shows how to decompose the first term; the same method applies

to the second term. Counting the free variables enclosed in the innermost max operator, we get five:  $i, k, u, v_1$ , and  $u_2$ . The decomposition eliminates one free variable,  $v_1$ . In the outermost level, there are six free variables left. The maximum number of interacting variables is six overall. So, we reduced the complexity of ITG decoding using bigram language model from  $O(n^7)$  to  $O(n^6)$ .



The hooks  $i$   $k$  that we have built for decoding with a bigram language model turn out to be similar to the hooks for bilexical parsing if we focus on the two boundary words  $v_1$  and  $u_2$  (or  $v_2$  and  $u_1$ )

$$\beta(X[i, j, u, v]) = \max \left\{ \begin{array}{l} \max_{k, v_1, u_2, Y, Z} \left[ \beta(Y[i, k, u, v_1]) \cdot \beta(Z[k, j, u_2, v]) \right. \\ \left. \cdot P(X \rightarrow [YZ]) \cdot \text{bigram}(v_1, u_2) \right], \\ \max_{k, v_2, u_1, Y, Z} \left[ \beta(Y[i, k, u_1, v]) \cdot \beta(Z[k, j, u, v_2]) \right. \\ \left. \cdot P(X \rightarrow \langle YZ \rangle) \cdot \text{bigram}(v_2, u_1) \right] \end{array} \right\} \quad (2)$$

$$\begin{aligned} & \max_{k, v_1, u_2, Y, Z} \left[ \beta(Y[i, k, u, v_1]) \cdot \beta(Z[k, j, u_2, v]) \cdot P(X \rightarrow [YZ]) \cdot \text{bigram}(v_1, u_2) \right] \\ &= \max_{k, u_2, Z} \left[ \max_{v_1, Y} \left[ \beta(Y[i, k, u, v_1]) \cdot P(X \rightarrow [YZ]) \cdot \text{bigram}(v_1, u_2) \right] \cdot \beta(Z[k, j, u_2, v]) \right] \end{aligned}$$

Figure 3: Equation for ITG decoding (top), with an efficient factorization (bottom)

that are interacting between two adjacent decoding constituents and relate them with the  $h'$  and  $h$  that are interacting in bilexical parsing. In terms of algebraic manipulation, we are also rearranging three factors (ignoring the non-lexical rules), trying to reduce the maximum number of interacting variables in any computation step.

#### 4.1 Generalization to $m$ -gram Cases

In this section, we will demonstrate how to use the hook trick for trigram decoding which leads us to a general hook trick for any  $m$ -gram decoding case.

We will work only on straight rules and use icons of constituents and hooks to make the equations easier to interpret.

The straightforward dynamic programming equation is:

$$\begin{array}{c} X \\ u1u2 \quad v1v2 \\ \boxed{\phantom{X}} \\ i \qquad j \end{array} = \max_{\substack{v_{11}, v_{12}, u_{21}, u_{22}, \\ k, Y, Z}} \begin{array}{c} [Y \quad X \quad Z] \\ u1u2 \quad v11v12 \quad u21u22 \quad v1v2 \\ \boxed{\phantom{X}} \quad \boxed{\phantom{X}} \\ i \qquad k \qquad j \end{array} \quad (3)$$

By counting the variables that are dependent on input sentence length on the right hand side of the equation, we know that the straightforward algorithm's complexity is  $O(n^{11})$ . The maximization computation is over four factors that are dependent on  $n$ :  $\beta(Y[i, k, u_1, u_2, v_{11}, v_{12}])$ ,  $\beta(Z[k, j, u_{21}, u_{22}, v_1, v_2])$ ,  $\text{trigram}(v_{11}, v_{12}, u_{21})$ , and  $\text{trigram}(v_{12}, u_{21}, u_{22})$ . As before, our goal is to cleverly bracket the factors.

By bracketing  $\text{trigram}(v_{11}, v_{12}, u_{21})$  and  $\beta(Y[i, k, u_1, u_2, v_{11}, v_{12}])$  together and maximizing over  $v_{11}$  and  $Y$ , we can build the the level-1 hook:

$$\begin{array}{c} [ \quad X \quad ] \\ u1u2 \quad v12 \quad u21 \\ \boxed{\phantom{X}} \\ i \qquad k \end{array} = \max_{v_{11}, Y} \begin{array}{c} [Y \quad X \quad Z] \\ u1u2 \quad v11v12 \quad u21 \\ \boxed{\phantom{X}} \\ i \qquad k \end{array}$$

The complexity is  $O(n^7)$ .

Grouping the level-1 hook and  $\text{trigram}(v_{12}, u_{21}, u_{22})$ , maximizing over  $v_{12}$ , we can build the level-2 hook:

$$\begin{array}{c} [ \quad X \quad ] \\ u1u2 \quad u21 \quad u22 \\ \boxed{\phantom{X}} \\ i \qquad k \end{array} = \max_{v_{12}} \begin{array}{c} [ \quad X \quad ] \\ u1u2 \quad v12 \quad u21 \quad u22 \\ \boxed{\phantom{X}} \\ i \qquad k \end{array}$$

The complexity is  $O(n^7)$ . Finally, we can use the level-2 hook to combine with  $Z[k, j, u_{21}, u_{22}, v_1, v_2]$  to build  $X[i, j, u_1, u_2, v_1, v_2]$ . The complexity is  $O(n^9)$  after reducing  $v_{11}$  and  $v_{12}$  in the first two steps.

$$\begin{array}{c} X \\ u1u2 \quad v1v2 \\ \boxed{\phantom{X}} \\ i \qquad j \end{array} = \max_{u_{21}, u_{22}, k, Z} \begin{array}{c} [ \quad X \quad ] \\ u1u2 \quad u21u22 \quad v1v2 \\ \boxed{\phantom{X}} \quad \boxed{\phantom{X}} \\ i \qquad k \qquad j \end{array} \quad (4)$$

Using the hook trick, we have reduced the complexity of ITG decoding using bigrams from  $O(n^7)$  to  $O(n^6)$ , and from  $O(n^{11})$  to  $O(n^9)$  for trigram

case. We conclude that for  $m$ -gram decoding of ITG, the hook trick can change the the time complexity from  $O(n^{3+4(m-1)})$  to  $O(n^{3+3(m-1)})$ . To get an intuition of the reduction, we can compare Equation 3 with Equation 4. The variables  $v_{11}$  and  $v_{12}$  in Equation 3, which are independent of  $v_1$  and  $v_2$  for maximizing the product have been concealed under the level-2 hook in Equation 4. In general, by building  $m - 1$  intermediate hooks, we can reduce  $m - 1$  free variables in the final combination step, hence having the reduction from  $4(m - 1)$  to  $3(m - 1)$ .

## 5 Generalization to Non-binary Bitext Grammars

Although we have presented our algorithm as a decoder for the binary-branching case of Inversion Transduction Grammar, the same factorization technique can be applied to more complex synchronous grammars. In this general case, items in the dynamic programming chart may need to represent non-contiguous span in either the input or output language. Because synchronous grammars with increasing numbers of children on the right hand side of each production form an infinite, non-collapsing hierarchy, there is no upper bound on the number of discontinuous spans that may need to be represented (Aho and Ullman, 1972). One can, however, choose to factor the grammar into binary branching rules in one of the two languages, meaning that discontinuous spans will only be necessary in the other language.

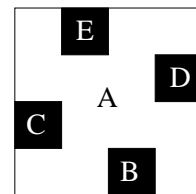
If we assume  $m$  is larger than 2, it is likely that the language model combinations dominate computation. In this case, it is advantageous to factor the grammar in order to make it binary in the output language, meaning that the subrules will only need to represent adjacent spans in the output language. Then the hook technique will work in the same way, yielding  $O(n^{2(m-1)})$  distinct types of items with respect to language model state, and  $3(m - 1)$  free indices to enumerate when combining a hook with a complete constituent to build a new item. However, a larger number of indices pointing into the input language will be needed now that items can cover discontinuous spans. If the grammar factorization yields rules with at most  $R$  spans

in the input language, there may be  $O(n^{2R})$  distinct types of chart items with respect to the input language, because each span has an index for its beginning and ending points in the input sentence. Now the upper bound of the number of free indices with respect to the input language is  $2R + 1$ , because otherwise if one rule needs  $2R + 2$  indices, say  $i_1, \dots, i_{2R+2}$ , then there are  $R + 1$  spans  $(i_1, i_2), \dots, (i_{2R+1}, i_{2R+2})$ , which contradicts the above assumption. Thus the time complexity at the input language side is  $O(n^{2R+1})$ , yielding a total algorithmic complexity of  $O(n^{3(m-1)+(2R+1)})$ .

To be more concrete, we will work through a 4-ary translation rule, using a bigram language model. The standard DP equation is:

$$\begin{array}{c} \text{A} \\ \boxed{u \quad v} \\ \text{i} \quad \text{j} \end{array} = \max_{\substack{v_3, u_1, v_1, u_4, v_4, u_2, \\ k_1, k_2, k_3, \\ B, C, D, E}} \begin{array}{c} \text{B} \quad \text{C} \quad \text{A} \quad \text{D} \quad \text{E} \\ \boxed{u \quad v_3 \quad u_1 \quad v_1 \quad u_4 \quad v_4 \quad u_2 \quad v} \\ \text{i} \quad \text{k}_1 \quad \text{k}_2 \quad \text{k}_3 \quad \text{j} \end{array} \quad (5)$$

This 4-ary rule is a representative difficult case. The underlying alignment pattern for this rule is as follows:



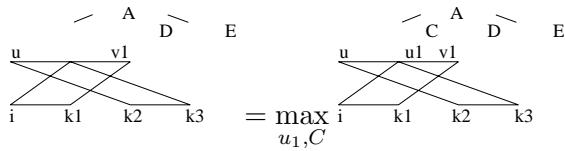
It is a rule that cannot be binarized in the bitext space using ITG rules. We can only binarize it in one dimension and leave the other dimension having discontinuous spans. Without applying binarization and hook trick, decoding parsing with it according to Equation 5 requires time complexity of  $O(n^{13})$ .

However, we can build the following partial constituents and hooks to do the combination gradually.

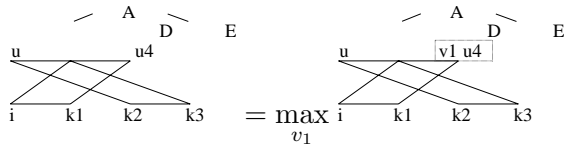
The first step finishes a hook by consuming one bigram. Its time complexity is  $O(n^5)$ :

$$\begin{array}{c} \text{C} \quad \text{A} \quad \text{D} \quad \text{E} \\ \boxed{u \quad u_1} \\ \text{k}_2 \quad \text{k}_3 \end{array} = \max_{v_3, B} \begin{array}{c} \text{B} \quad \text{C} \quad \text{A} \quad \text{D} \quad \text{E} \\ \boxed{u \quad v_3 \quad u_1} \\ \text{k}_2 \quad \text{k}_3 \end{array}$$

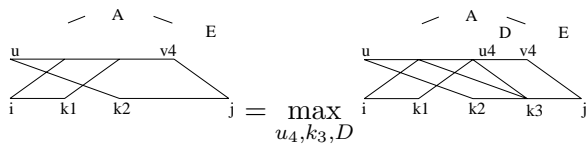
The second step utilizes the hook we just built and builds a partial constituent. The time complexity is  $O(n^7)$ :



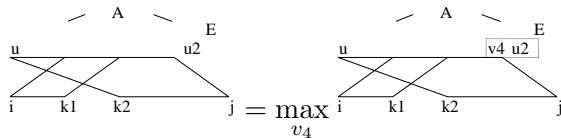
By “eating” another bigram, we build the second hook using  $O(n^7)$ :



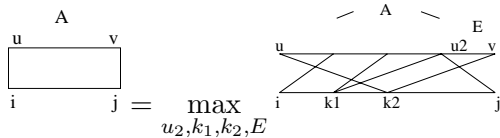
We use the last hook. This step has higher complexity:  $O(n^8)$ :



The last bigram involved in the 4-ary rule is completed and leads to the third hook, with time complexity of  $O(n^7)$ :



The final combination is  $O(n^7)$ :

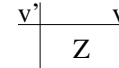


The overall complexity has been reduced to  $O(n^8)$  after using binarization on the output side and using the hook trick all the way to the end. The result is one instance of our general analysis: here  $R = 2$ ,  $m = 2$ , and  $3(m - 1) + (2R + 1) = 8$ .

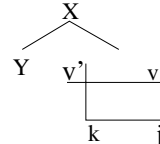
## 6 Implementation

The implementation of the hook trick in a practical decoder is complicated by the interaction with

pruning. If we build hooks looking for all words in the vocabulary whenever a complete constituent is added to the chart, we will build many hooks that are never used, because partial hypotheses with many of the boundary words specified by the hooks may never be constructed due to pruning. Instead of actively building hooks, which are intermediate results, we can build them only when we need them and then cache them for future use. To make this idea concrete, we sketch the code for bigram integrated decoding using ITG as in Algorithm 1. It is worthy of noting that for clarity we



are building hooks in shape of  $\begin{matrix} v & v \\ & Z \\ k & j \end{matrix}$ , instead



of  $\begin{matrix} k & j \end{matrix}$  as we have been showing in the previous sections. That is, the probability for the grammar rule is multiplied in when a complete constituent is built, rather than when a hook is created. If we choose the original representation, we would have to create both straight hooks and inverted hooks because the straight rules and inverted rules are to be merged with the “core” hooks, creating more specified hooks.

## 7 Conclusion

By showing the parallels between lexicalization for language model state and lexicalization for syntactic heads, we have demonstrated more efficient algorithms for previously described models of machine translation. Decoding for Inversion Transduction Grammar with a bigram language model can be done in  $O(n^6)$  time. This is the same complexity as the ITG alignment algorithm used by Wu (1997) and others, meaning complete Viterbi decoding is possible without pruning for realistic-length sentences. More generally, ITG with an  $m$ -gram language model is  $O(n^{3+3(m-1)})$ , and a synchronous context-free grammar with at most  $R$  spans in the input language is  $O(n^{3(m-1)+(2R+1)})$ . While this improves on previous algorithms, the degree in  $n$  is probably still too high for complete search to be practical with such models. The interaction of the hook technique with pruning is an interesting

---

**Algorithm 1** ITGDecode( $N_t$ )

---

**for all**  $s, t$  such that  $0 \leq s < t \leq N_t$  **do**  
  **for all**  $S$  such that  $s < S < t$  **do**  
    ▷ straight rule  
    **for all** rules  $X \rightarrow [YZ] \in G$  **do**  
      **for all**  $(Y, u_1, v_1)$  possible for the span of  $(s, S)$  **do**  
        ▷ a hook who is on  $(S, t)$ , nonterminal as  $Z$ , and outside expectation being  $v_1$  is required  
        **if not** *exist\_hooks* $(S, t, Z, v_1)$  **then**  
          *build\_hooks* $(S, t, Z, v_1)$   
        **end if**  
        **for all**  $v_2$  possible for the hooks in  $(S, t, Z, v_1)$  **do**  
          ▷ combining a hook and a hypothesis, using straight rule  
           $\beta(s, t, X, u_1, v_2) =$   
           $\max \left\{ \beta(s, t, X, u_1, v_2), \beta(s, S, Y, u_1, v_1) \cdot \beta^+(S, t, Z, v_1, v_2) \cdot P(X \rightarrow [YZ]) \right\}$   
        **end for**  
      **end for**  
    **end for**  
    ▷ inverted rule  
    **for all** rules  $X \rightarrow \langle YZ \rangle \in G$  **do**  
      **for all**  $(Z, u_2, v_2)$  possible for the span of  $(S, t)$  **do**  
        ▷ a hook who is on  $(s, S)$ , nonterminal as  $Y$ , and outside expectation being  $v_2$  is required  
        **if not** *exist\_hooks* $(s, S, Y, v_2)$  **then**  
          *build\_hooks* $(s, S, Y, v_2)$   
        **end if**  
        **for all**  $v_1$  possible for the hooks in  $(s, S, Y, v_2)$  **do**  
          ▷ combining a hook and a hypothesis, using inverted rule  
           $\beta(s, t, X, u_2, v_1) =$   
           $\max \left\{ \beta(s, t, X, u_2, v_1), \beta(S, t, Z, u_2, v_2) \cdot \beta^+(s, S, Y, v_2, v_1) \cdot P(X \rightarrow \langle YZ \rangle) \right\}$   
        **end for**  
      **end for**  
    **end for**  
  **end for**  
**end for**

**routine** *build\_hooks* $(s, t, X, v')$

**for all**  $(X, u, v)$  possible for the span of  $(s, t)$  **do**  
  ▷ combining a bigram with a hypothesis  
   $\beta^+(s, t, X, v', v) =$   
   $\max \left\{ \beta^+(s, t, X, v', v), \text{bigram}(v', u) \cdot \beta(s, t, X, u, v) \right\}$   
**end for**

---

area for future work. Building the chart items with hooks may take more time than it saves if many of the hooks are never combined with complete constituents due to aggressive pruning. However, it may be possible to look at the contents of the chart in order to build only those hooks which are likely to be useful.

## References

- Aho, Albert V. and Jeffery D. Ullman. 1972. *The Theory of Parsing, Translation, and Compiling*, volume 1. Englewood Cliffs, NJ: Prentice-Hall.
- Charniak, Eugene. 1997. Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pages 598–603, Menlo Park, August. AAAI Press.
- Collins, Michael. 1997. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Conference of the Association for Computational Linguistics (ACL-97)*, pages 16–23, Madrid, Spain.
- Eisner, Jason and Giorgio Satta. 1999. Efficient parsing for bilexical context-free grammars and head automaton grammars. In *37th Annual Meeting of the Association for Computational Linguistics*.
- Melamed, I. Dan. 2003. Multitext grammars and synchronous parsers. In *Proceedings of the 2003 Meeting of the North American chapter of the Association for Computational Linguistics (NAACL-03)*, Edmonton.
- Wu, Dekai. 1996. A polynomial-time algorithm for statistical machine translation. In *34th Annual Meeting of the Association for Computational Linguistics*.
- Wu, Dekai. 1997. Stochastic inversion transduction grammars and bilingual parsing of parallel corpora. *Computational Linguistics*, 23(3):377–403.