

# (Bi-)Lexicalized Context-Free Grammars

Christoph Dörr

University of Saarbrücken  
[Chris.Doerr@gmx.de](mailto:Chris.Doerr@gmx.de)

## Abstract

Several recent stochastic parsers use bilexicalized grammars, where each word idiosyncratically prefers particular complements with particular head words. These parsers' complexity is  $O(n^5)$ . For two different bilexicalized formalisms, I present an algorithm improving this upper bound to  $O(n^4)$ . For a common special case, even this can be reduced to  $O(n^3)$ . I'll present a  $O(n^3)$  algorithm with an improved grammar constant.

## 1. Introduction

Lexicalized grammars specify syntactic facts about each word of a language - in particular the type of arguments that a word can or must take. Early mechanisms satisfying this requirement utilize categorical grammar (Bar-Hillel, 1953) and subcategorization frames (Chomsky, 1965). Other lexicalized formalisms are (Schabes et al., 1988; Mel'cuk 1988, Pollard and Sag, 1994).

But instead of just specifying a possible argument for a word, especially for natural language, it is advantageous to specify possible head words for those arguments. For instance, “solved” requires a NP object. But some of the NP objects are semantically or lexically more appropriate here than others, and the appropriateness depends largely on the NP's head (e.g. “puzzle”). The acceptability of “Nora solved a puzzle” then depends on the grammar writer's assessment whether a puzzle can be solved. Grammars that record such facts are called “bilexical”. Thus, a bilexical grammar makes many stipulations about compatibility of particular pairs of words in particular roles. Formalisms describing lexical grammars, as well as such describing bilexical grammars are of high theoretical and practical interest to the computational linguistics community. Using probabilistic or weighted versions of bilexical grammars, recent real-world parsers have improved state-of-the-art parsing accuracy (Alshawi, 1996; Eisner, 1996; Charniak, 1997; Collins, 1997). All parsers have in common that soft selectional restrictions are used to solve ambiguities.<sup>1</sup> But even with excessive

---

<sup>1</sup> Other relevant parsers simultaneously consider two or more words that are not necessarily in a dependency relationship (Lafferty et al., 1992; Magermann, 1995)

pruning, all above algorithms run in time  $O(n^5)$ . The reason is that bilexical grammars are huge. Notice that the part of the grammar relevant to an input string of size  $n$  has size  $O(n^2)$  in practice! However, in this paper I show that it's possible to improve the complexity

- for bilexicalized context-free grammars to  $O(n^4)$ .
- for head-automaton grammars to  $O(n^4)$ .
- for a common special case of these grammars to  $O(n^3)$ , improving the grammar constant of the  $O(n^3)$  algorithm presented by (Eisner, 1997)

The presented algorithms propose new kinds of subderivations that are not constituents and are assembled into full constituents using dynamic programming. This paper is based on (Eisner and Satta, 1999).

## 2. Notation of context-free grammars and naming conventions

I assume that the reader is familiar with context-free grammars. In this paper I'll follow the notation that is proposed by (Harrison, 1978; Hopcroft and Ullman, 1979). A CFG is a 4-tuple  $G = (V_N, V_T, P, S)$ , where  $V_N$  and  $V_T$  are finite, disjoint sets of nonterminal and terminal symbols, respectively, and  $S \in V_N$ .  $P$  is a set of productions, where each production has the form  $A \rightarrow \alpha$ , where  $A \in V_N$ ,  $\alpha \in (V_N \cup V_T)^*$ . A grammar is in Chomsky normal form (CNF), if each production in  $P$  has the form  $A \rightarrow BC$  or  $A \rightarrow a$ , for  $A, B, C \in V_N$  and  $a \in V_T$ .<sup>2</sup> Throughout this paper I'll use the following naming conventions:

- $a, b, c, d$  denote symbols in  $V_T$
- $A, B, C$  denote symbols in  $V_N$
- $w, x, y$  denote strings in  $V_T^*$
- $\alpha, \beta, \gamma, \dots$  denote strings in  $(V_N \cup V_T)^*$
- $w = d_1 d_2 \dots d_n$  denotes the input string given to the parser (together with an CFG  $G$ )
- $h, i, j, k$  are positive integers, assumed to be  $\leq n$  when we use them as indices into  $w$
- $w_{i,j}$  denotes the input substring  $d_i \dots d_j$  ( $w_{i,j} = \epsilon$  if  $i > j$ )

Finally we define the derivation relation  $\Rightarrow$ , its transitive and reflexive closure  $\Rightarrow^*$  and the language  $L(G)$  generated by an CFG  $G$  as usual. We write  $\alpha \underline{\beta} \delta \Rightarrow^* \alpha \gamma \beta$  for a derivation in which only  $\beta$  is rewritten.

---

<sup>2</sup> (Notice that  $S \rightarrow \epsilon$  is in CNF, if  $S$  does not appear on the right hand side of any of the productions. But  $S \rightarrow \epsilon$  is not allowed in our bilexicalized CFG, as for every rule in  $P$ , each nonterminal must be lexicalized by a head word.)

### 3. Probabilistic Context-Free Grammars

To get an better idea of the advantages of BCFGs we first have a look at PCFGs, recently the formalism of choice for many researchers in computational linguistics. A PCFG is a CFG as defined before, which, in addition, defines a probability function  $F$  that assigns a probability to each production in  $P$ , such that  $\sum_{\alpha} F(A \rightarrow \alpha) = 1$ .

$S \rightarrow NP VP$	1.0	$NP \rightarrow NP PP$	0.4
$PP \rightarrow P NP$	1.0	$NP \rightarrow \textit{astronomers}$	0.1
$VP \rightarrow V NP$	0.64	$NP \rightarrow \textit{eyes}$	0.8
$VP \rightarrow VP PP$	0.36	$NP \rightarrow \textit{telescopes}$	0.1
$P \rightarrow \textit{with}$	1.0	$NP \rightarrow \textit{saw}$	0.04
$V \rightarrow \textit{saw}$	1.0	$NP \rightarrow \textit{stars}$	0.18

Fig.1: The productions and probabilities for the sample PCFG

I.e. the sum of the probabilities of all productions containing  $A$  as the parent is 1. With these probabilities, the probability of a parse is the product of the probabilities of the used productions. Consider the simple PCFG in Fig.1 and the input string “Astronomers saw stars with eyes”. Fig.2 and Fig.3 give two different parse trees with the probability of the corresponding parse. Although one parse is semantically much better than the other, both parses have almost the same probability and thus there is no way for us to decide which one to prefer. The crucial point in the derivation is the question which of the rules  $VP \rightarrow VP PP$  and  $NP \rightarrow NP PP$  is better. This cannot generally be said for these rules. What we would need here to solve the ambiguity is the possibility to see the head words of the prepositional phrase (PP), of the verb phrase (VP) and of the noun phrase (NP), i.e. the words the currently parsed substrings refer to. This would allow us to see that a VP headed by *see* is more likely to be modified by a PP headed by *eyes* than by a NP headed by *stars*. And this is exactly what bilexicalized grammars do! Now, as we have a better intuition of the usefulness of BCFGs, it is convenient to consider CFGs instead of PCFGs in the upcoming chapters. CFGs are more lightweight and all results can straightforwardly be applied to PCFGs.

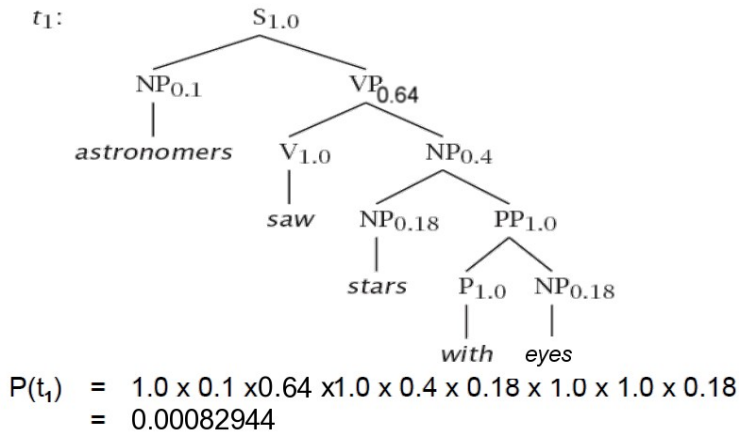


Fig.2: The first possible parse

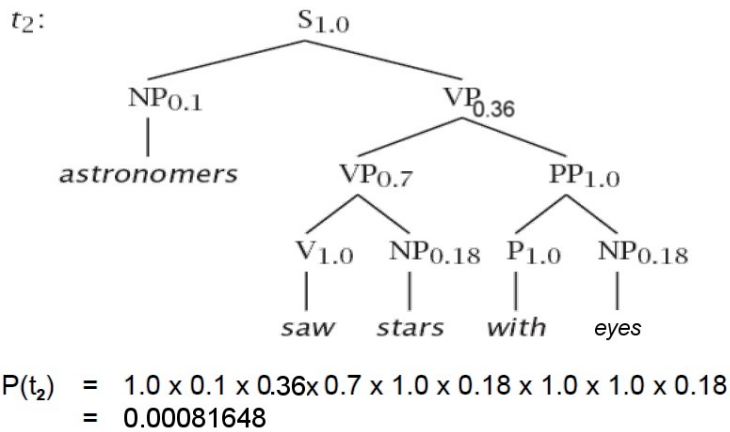


Fig.3: The second possible parse

#### 4. Bilexical context-free grammars

After discussing the motivation for bilexicalized grammars, I'll define a formalism that satisfies the requirements established in §3. That is, a formalism that captures the dependencies among pairs of words in  $V_T$ . Let  $G$  be a **bilexical CFG** with  $G = \{V_N, V_T, P, T[\$]\}$ . Let  $V_D$  be a set of “delexicalized nonterminals”, i.e. “traditional” nonterminals as defined in §2. Then  $V_N = \{A[a] : A \in V_D, a \in V_T\}$ . As the start symbol  $T[\$]$  is an element of  $V_N$ , it is necessarily a lexicalized “traditional” nonterminal too. Its **lexical head**  $\$$  is a terminal appearing in every sentence of  $L(G)$ . In most cases it is convenient to consider the language  $L(G) = \{x : x\$ \in L(G)\}$ . Finally, each production

in  $P$  has the form

- $A[a] \rightarrow B[b] C[a]$
- $A[a] \rightarrow C[a] B[b]$
- $A[a] \rightarrow a$

In each rule the lexical head  $a$  on the left is inherited from the constituent's **head child** in the parse tree. With such a grammar we're able to encode lexically specific preferences as required in §2. Let's observe this formalism at work. Consider the example with the astronomers above.  $V_T$ ,  $V_N$  and  $T[\$]$  are defined straightforwardly.  $P$  contains the following productions.

$S[\text{saw}] \rightarrow \text{NP}[\text{astronomers}] \text{VP}[\text{saw}]$   
 $\text{VP}[\text{saw}] \rightarrow \text{V}[\text{saw}] \text{NP}[\text{stars}]$   
 $\text{PP}[\text{eyes}] \rightarrow \text{P}[\text{with}] \text{NP}[\text{eyes}]$   
 $\text{VP}[\text{saw}] \rightarrow \text{VP}[\text{saw}] \text{PP}[\text{eyes}]$   
 $\text{NP}[\text{stars}] \rightarrow \text{NP}[\text{stars}] \text{PP}[\text{eyes}]$

As can easily be seen, in a bilexicalized CFG such as this, we can selectively rule out parses where *stars* is modified by a PP with head word *eyes* by simply leaving out the rule  $\text{NP}[\text{stars}] \rightarrow \text{NP}[\text{stars}] \text{PP}[\text{eyes}]$  (or giving the rule a lower probability in a bilexicalized PCFG). Unfortunately this increased expressiveness has a backdraw: such grammars are huge. Consider a CKY-based parser with the running time  $O(n^3 * |P|)$  (Younger, 1967; Aho and Ullman, 1972). In the worst case, for CFGs we have  $|P| = |V_N|$  and for BCFGs even  $|P| = |V_D|^3 * |V_T|^2$ . Thus  $|P|$  depends on the vocabulary size, which is very large for natural languages. Observing that there are at most  $n$  words of the whole vocabulary relevant for a given input string  $w = d_1 d_2 \dots d_n$ , we can rewrite  $|P| = |V_D|^3 * |V_T|^2$  to  $|P| = |V_D|^3 * \min(|V_T|, n)^2$  and finally, as in practical applications,  $n \ll V_T$ ,  $|P| = |V_D|^3 * n^2$ . With these considerations, the running time of the parser is  $O(n^3 * |V_N|^3)$  for CFGs and  $O(n^3 * |V_D|^3 * n^2) = O(n^5 * |V_D|^3)$  for BCFGs. Thus, parsing is a factor of  $n^2$  slower for BCFG than for CFG! To get a better understanding for the improved algorithm I'll give in the next section, we'll have a closer look at the source of this. The standard CKY algorithm contains derivations of the following form:

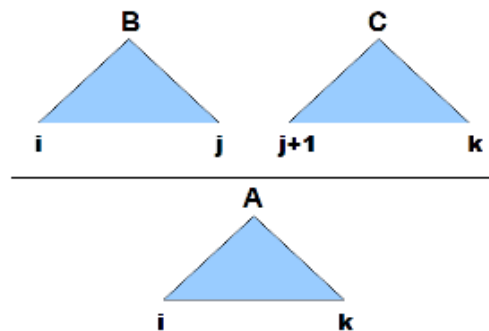


Fig.4: The "standard" derivation for CKY for CFGs

We have three variables ranging over  $n$  that can be combined in  $n^3$  different ways. And indeed, this is the  $n^3$  we find in the worst case running time for standard CKY for CFGs. But for BCFGs we have derivations of the form in Fig.5, reflecting the fact that we have for each production two additional variables to consider, namely the two lexical heads. And again, each variable ranges over  $n$ , resulting in  $n^5$  different combinations. Obviously the key to an improved running time is decreasing the number of variables considered at once. And indeed, this is possible, as I'll show in the next sections.

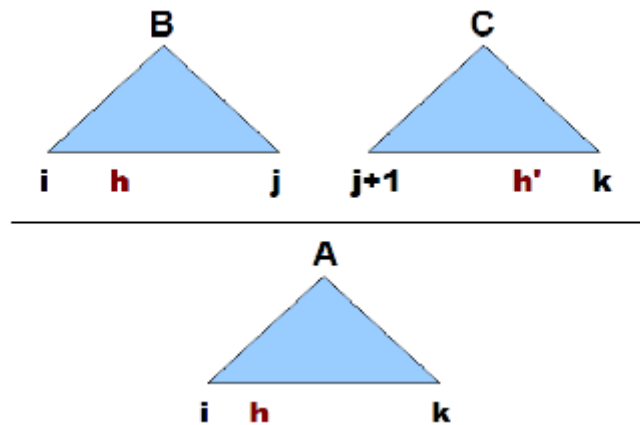


Fig.5: The “standard derivation” for CKY for BCFG

## 5. Parsing BCFGs in $O(n^4)$

Take a look at Fig.5. What  $C$  can be combined with a given  $B$ ? To find a suitable  $C$ , you have to vary the width, i.e. the “end-point”  $k$ , and the lexical head  $h'$ . If you vary  $k$  and  $h'$  at the same time, you get  $n^2$  possibilities resulting in  $O(n^2)$  running time. The idea behind the  $O(n^4)$  algorithm is to split this step up into two single steps. First combine  $B$  just with all possible head words  $h'$  for  $C$ . This is legal if there's a production in  $P$  where  $h'$  (on the right side) is a parent of  $h$  (on the left side), i.e.  $A[h] \rightarrow B[h] C[h']$ . This results in a structure that categorical grammar would call an  $A/C$ . This is an  $A$  missing its head child on the right, a  $C$  headed at position  $h$ . The derivation for this step is shown in Fig.6.

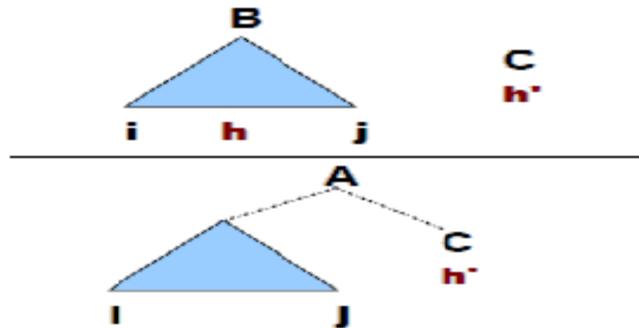


Fig. 6: The first step of the algorithm sketch. Combining B with the disembodied C headed at  $h'$

In the next step we vary  $k$ , i.e. the width of  $C$ , where the  $C$  is headed at  $h'$  and starts at  $j+1$ . Each of the resulting  $C$ 's can be combined with the  $A/C$  to get the “full”  $A$ . This step is shown in Fig.7.

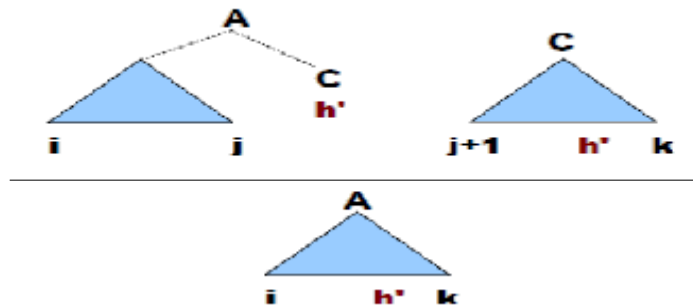


Fig. 7: The second step of the algorithm sketch: combining  $A/C$  with the body of  $C$  headed at  $h'$ .

And again we have only 4 variables we range over, resulting in a running time of  $O(n^4)$ . And the combination of these two steps runs in time  $O(2*n^4) = O(n^4)$ . The formal definition of the algorithm is given in the appendix from (Eisner and Satta, 1999)

## 6. Head automaton grammar

It's even possible to improve the running time to  $O(n^3)$  for a common special case of BCFGs, so called split grammars (see §7). But to understand what split grammars are, it is convenient to define another formalisms, namely head automaton grammars (Alshawi, 1996), that allows an intuitive understanding of split grammars, and show that it's possible to parse a  $n$ -length input string for the general case of HAGs in  $O(n^4)$ . After that I'll show that it's easy to transform a HAG into a BCFG that constructs the same language.

A Head Automaton Grammar is a function that defines a Head Automaton (HA)  $H_a$  for each element  $a$  of its finite domain, i.e.  $V_T$ . An example should give us a first idea how a  $H_a$  works:

“[Good old **Peggy**]solved[the **puzzle**][**with** her teeth].”

The words in braces are already “processed” by their HAs. Notice that the input for  $H_{\text{puzzle}}$  was “the” on the left. “Her teeth” was already consumed by  $H_{\text{with}}$ . We'll only observe the HA for “solved” at work (Note that here the preposition is the head of a PP, contrary to what we did in the introductory example for PCFG above, where the noun was the head of the PP.):

<b>Peggy</b> <u>solved</u> [puzzle] [with]	(state = V)
<b>Peggy</b> <u>solved</u> [with]	(state = VP)
<b>Peggy</b> <u>solved</u>	(state = VP)
<u>solved</u>	(state = S;halt)

Let  $D = \{\rightarrow, \leftarrow\}$  and  $\$ \in V_T$  be a special start symbol. Then, for every  $a \in V_T$ ,

$H_a = \{Q_a, V_T, \delta_a, I_a, F_a\}$  where

- $Q_a$  is a finite set of states
- $I_a, F_a \subseteq Q_a$  are sets of initial and final states, respectively
- $\delta_a$  is a transition function mapping  $Q_a \times V_T \times D$  to  $2^{Q_a}$ , the power set of  $Q_a$

You can imagine a HA as an automaton  $H_a$ , whose head points between two words of the input string. Thus a HA is an acceptor for a pair of strings  $\langle z_l, z_r \rangle \in V_T^* \times V_T^*$ . For instance, if  $b$  is the leftmost symbol of  $z_r$ , the automaton is in state  $q$  and  $q' \in \delta_a(q, b, \rightarrow)$ , HA can move from  $q$  to  $q'$ , removing  $b$  from the left end of  $z_r$ . The input string is accepted if a state  $q'' \in F_a$  is reached when  $z_r, z_l = \epsilon$ .

The derivation relation  $\vdash_a$  is defined for each head automaton  $H_a$  (similar to  $\Rightarrow$  for CFG) as a binary relation on  $Q_a \times V_T^* \times V_T^*$  as follows:

For every  $q \in Q_a, x, y \in V_T^*, b \in V_T, d \in D$  and  $q' \in \delta_a(q, b, d)$

- $(q, xb, y) \vdash_a (q', x, y)$  if  $d = \leftarrow$
- $(q, x, by) \vdash_a (q', x, y)$  if  $d = \rightarrow$



The reflexive, transitive closure of  $\vdash_a$  is written  $\vdash_a^*$ . The language generated by  $H_a$  is the set

$$L(H_a) = \{\langle z_l, z_r \rangle \mid (q, z_l, z_r) \vdash_a^*(r, \varepsilon, \varepsilon), q \in I_a, r \in F_a\}$$

The language  $L_S$  generated by the entire grammar  $H$  is now defined as follows.

Given  $H$ , we define  $L_a$  for all  $a \in V_T$  to be the least set such that if  $\langle x, y \rangle \in L(H_a)$  and  $x' \in L_x, y' \in L_y$ , then  $x'ay' \in L_a$ .

To conclude the formal definition of HAGs, let's see a simple one in practice. I'll (informally) define a HAG  $H$  that generates only the two sentences "The woman sleeps soundly \$" and "A woman sleeps soundly \$" and show how the language of  $H$  is built up.  $H$  contains the following HAs:

- $H_{\text{the}}: \delta_{\text{the}} = \emptyset, L(H_{\text{the}}) = \{\langle \varepsilon, \varepsilon \rangle\}, L_{\text{the}} = \{\text{the}\}$
- $H_a: \delta_a = \emptyset, L(H_a) = \{\langle \varepsilon, \varepsilon \rangle\}, L_a = \{a\}$
- $H_{\text{woman}}: \delta_{\text{woman}} = \{(q, \text{the}, \leftarrow, q')\}, L(H_{\text{woman}}) = \{\langle \text{the}, \varepsilon \rangle, \langle a, \varepsilon \rangle\},$   
 $L_{\text{woman}} = \{\text{the woman, a woman}\}$
- $H_{\text{soundly}}: \delta_{\text{soundly}} = \emptyset, L(H_{\text{soundly}}) = \{\langle \varepsilon, \varepsilon \rangle\}, L_{\text{soundly}} = \{\text{soundly}\}$
- $H_{\text{sleeps}}: \delta_{\text{sleeps}} = \{(q, \text{woman}, \leftarrow, q'), (q', \text{soundly}, \rightarrow, q'')\}, L(H_{\text{sleeps}}) = \{\langle \text{woman}, \text{soundly} \rangle\}$   
 $L_{\text{sleeps}} = \{\text{the woman sleeps soundly, a woman sleeps soundly}\}$
- $H_S: \delta_S = \{(q, \text{sleeps}, \leftarrow, q')\}, L(H_S) = \{\langle \text{sleeps}, \varepsilon \rangle\},$   
 $L_S = \{\text{the woman sleeps soundly $, a woman sleeps soundly $}\}$

Now I'll give a simple algorithm transforming a HAG  $H$  into a BCFG  $G$  generating the same language. Even more, the BCFG preserves derivation ambiguities. I.e. if there is more than one derivation for an input string, then there are equally many different derivations for this input string in the BCFG. If not stated otherwise, variables have the same meaning as above.

Let  $V_D$  be an arbitrary set of size  $t = \max\{|Q_a| : a \in V_T\}$ , and for each  $a$ , define an arbitrary injection  $f_a : Q_a \rightarrow V_D$ . Now  $G = (V_N, V_T, P, T[\$])$  is constructed as follows:

- (i)  $V_N = \{A[a] : A \in V_D, a \in V_T\}$  as usual for BCFG (see §4)
- (ii)  $P$  is the set of productions having one of the following forms, where  $a, b \in V_T$ :
  - $A[a] \rightarrow B[b] C[a]$ , where  $A = f_a(r), B = f_b(q'), C = f_a(q)$  for some  $q' \in I_b, q \in Q_a, r \in \delta_a(q, b, \leftarrow)$
  - $A[a] \rightarrow C[a] B[b]$ , where  $A = f_a(r), B = f_b(q'), C = f_a(q)$  for some  $q' \in I_b, q \in Q_a, r \in \delta_a(q, b, \rightarrow)$
  - $A[a] \rightarrow a$ , where  $A = f_a(q)$  for some  $q \in F_a$
- (iii)  $T = f_S(q)$ , where we assume w.l.o.g that  $I_S$  is a singleton set  $\{q\}$

To show that  $G$  and  $H$  generate the same languages it would be necessary to give a formal proof that they admit isomorphic derivations. But as this is neither straightforward nor needed for the understanding of the further discussion, this proof is omitted here and we just observe that if  $\langle x, y \rangle = \langle b_1 b_2 \dots b_j, b_{j+1} \dots b_k \rangle \in L(H_a)$  then

$A[a] \Rightarrow^* B_1[b_1] \dots B_j[b_j] a B_{j+1}[b_{j+1}] \dots B_k[b_k]$ , for any  $A, B_1, \dots, B_k$  that map to initial states in  $H_a, H_{b_1}, \dots, H_{b_k}$ , respectively. This means that a HA can consume words adjacent to it on either side, but first these words must consume *their* dependents (with the HAs defined for these words in  $H$ ).

A special case are **split** head automata. A head automaton  $H_a$  is called split if it has no states that can be entered on a  $\leftarrow$  transition and exited on a  $\rightarrow$  transition. Such a HA accepts the input string  $\langle x, y \rangle$  by consuming all its right dependents first before consuming the left dependents. After reading  $y$ , the automaton is said to be in a flip state. A flip state allows entry on a  $\rightarrow$  transition and that either allows exit on a  $\leftarrow$  transition or is a final state. Obviously a split HA allows no cycles. We are interested in HAG that only consist of split HA. Such a “split HAG” corresponds to a BCFG in which any derivation  $A[a] \Rightarrow^* xay$  has the form  $A[a] \Rightarrow^* xB[a] \Rightarrow^* xay$ . HAs with cycles are only necessary for formal languages of the form  $\{\langle b^n, c^n \rangle : n \geq 0\}$ , where word  $a$  takes  $2n$  dependents. According to (Eisner and Satta, 1999), for natural language such a requirement is never needed (indeed, this is an informal claim). To conclude this chapter I follow (Eisner and Satta, 1999) in claiming that each HAG for natural languages can be transformed to a split HAG. This is important, as we'll see in §9.

## 7. Expressiveness of (split) bCFG

Let's have a closer look at how the expressiveness of (split) BCFGs and of CFGs are related.

- *BCFGs are strongly equivalent with CFGs.*  
Proof idea: Let the CFG be in CNF. For each  $a \in V_T$  add  $a$  as the head to each nonterminal in each rule, e.g.  $A \rightarrow BC \Rightarrow A[a] \rightarrow B[a]C[a]$ .
- *Split BCFGs are weakly equivalent with general BCFGs.*  
Proof idea: Given a BCFG, modify each production to specify the head of the left child as the lexical head. In the resulting grammar all dependents of a head are on its right. Thus the resulting grammar is split. Indeed, the new split grammar generates the same tree structures but different dependency structures as the given BCFG. I.e. the generated trees only differ in the labels of the nodes.

## 8. Split HAG and BCFG in time $O(n^3)$

I'll now give an algorithm that parses split HAGs (and hence BCFGs; see §6) in time  $O(n^3)$  (Eisner and Satta, 1999), improving the grammar constant of the  $O(n^3)$  algorithm presented in (Eisner, 1997). The reason why we're that interested in split grammars is that such grammars allow derivation rules as shown in Fig.8.

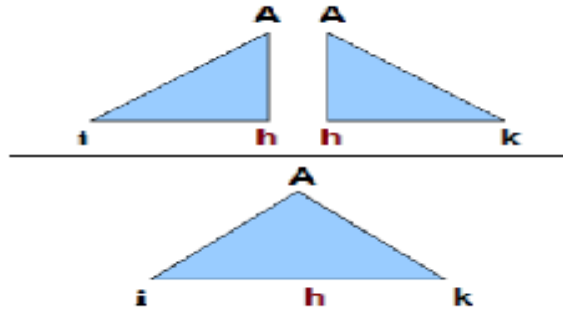


Fig. 8: Split grammars allow separate, simultaneous parsing of the left and right dependents

This rule says, that we can combine the left and right body of  $A[h]$ , after we have processed them separately, to get an "full"  $A[h]$ . This kind of derivation rule is crucial for our improved algorithm that is based on the same idea then the algorithm in §4. But this time, we split the varying of  $k$ , the end-point of  $C$ , and  $j$ , the end-point of  $B$ , also representing the midpoint between  $B$  and  $C$ . First, we vary  $j$ . To exclude  $k$  from this step, we make use of the derivation rule in Fig.8 and combine an  $B$  just with the left body of  $C$ . This results in an  $A$  that is missing the right body of  $C$ , headed at  $h'$  and of arbitrary width. Hence, we range over the 4 variables  $h, h', i$  and  $j$ , which needs  $O(n^4)$  time. The derivation rule for this step is given in Fig.9.

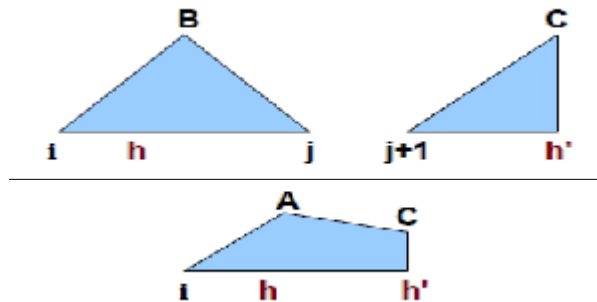


Fig. 9: First step: Combining B only with the left half of  $C[h']$

In the second step we construct the "right-body"  $C$ s fulfilling the above requirements to merge it with the "incomplete"  $A$  to a "full"  $A$  headed at  $h$ . To do so, we have to range over the 4 variables  $h, h', i$  and  $k$ . Again this needs  $O(n^4)$  time resulting in an overall running time of  $O(2n^4) = O(n^4)$ . The derivation rule for the second step is given in Fig.10.

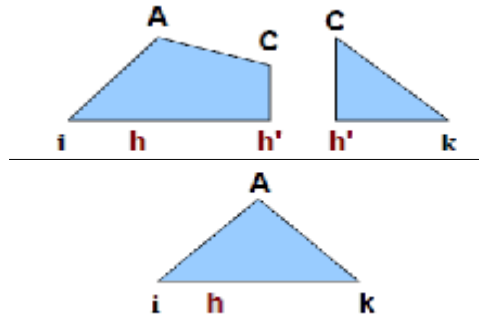


Fig. 10: The second step: Merging the A missing the right body of  $C[h']$  with the matching "right body"  $C[h']$

To get down to  $O(n^3)$  we apply this "trick" not only to the right body of  $C$ , but also to the left body of  $B$ . This results in a "right-body"  $A$  headed at  $h$ . Such  $A$ s are then combined with their "left-body"  $A$  counterpart to the "full"  $A$  in an additional step. This step just needs constant time and thus does not affect the running time. And because we even don't care for the starting-point of  $B$  "i" now, we only consider a maximum of three variables, namely  $h$ ,  $h'$  and  $j$ , at once, allowing us to run this part of the algorithm (the part that brought us to  $O(n^5)$ ) in time  $O(n^3)$ ! Fig.11 shows a derivation tree that illustrates this idea. Keep in mind that all this is possible because in split  $HA$ , one can separately find the half-path before the flip state (which accepts  $z_r$ ) and the half-path after the flip state (which accepts  $z_l$ ). To join these two half-paths into an accepting path they only have to share the same flip state  $s$ , i.e. one path starts where the other ends. Of course, to do this check we have to keep track of  $s$ . The formal definition of the algorithm based on this sketch is given in the appendix of (Eisner and Satta, 1999).

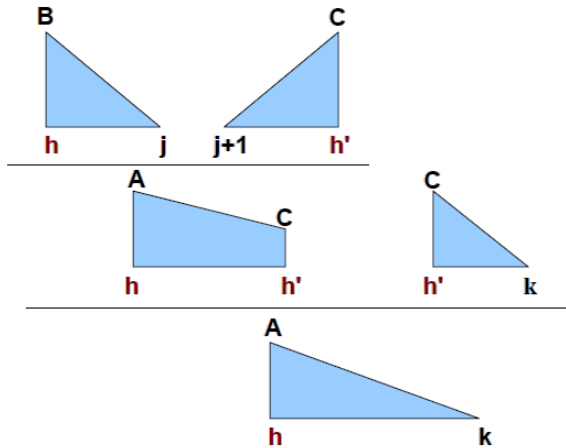


Fig.11: Ignoring the left part of the body of  $B$  it's even possible to get to  $O(n^3)$ !

## 9. Theoretical Speedup

With  $n$  = input string length,  $g$  = grammar constant (polysemy factor) and  $t$  = traditional nonterms or automaton states, the different considered algorithms in this paper have the following running times.

For the general case:

- Naive:  $O(n^5g^2t)$
- New (Eisner and Satta, 1999):  $O(n^4g^2t)$

For split grammars:

- Old (Eisner, 1997):  $O(n^3g^3t^2)$
- New (Eisner and Satta, 1999):  $O(n^3g^2t)$

In the worst case the polysemy factor  $g$  equals  $t$  for BCFGs. But in most cases it's smaller than  $t$ . More information on polysemy can, for instance, be found in (Eisner and Satta, 1997; Eisner, 2000). Notice that all running times are independent of the vocabulary size!

These theoretical speedups are indeed confirmed in practical applications. Compared with the Eisner (1996) Treebank WSJ parser and its split lexical grammar, a common  $O(n^5)$  parser, the new algorithm gives a 5x speedup with pruning and even a 19x speedup for exhaustive parsing!

## 10. Final remarks

I've described three different grammatical rewriting systems that encode dependencies between pairs of words, namely bilexical context-free grammars (Eisner, 1997; Eisner and Satta, 1999; Eisner, 2000), head automaton grammars (Alshawi, 1996) and split HAG (Eisner, 1997; Eisner and Satta 1999). Following (Eisner and Satta, 1999) I've shown that HAGs are isomorphic to BCFGs (Eisner and Satta, 1999). For each of these formalisms I've given an algorithm sketch improving the standard  $O(n^5)$  running time. For general BCFGs and HAGs the running time was improved to  $O(n^4)$  and for split HAGs the running time was even improved to  $O(n^3)$ , simultaneously decreasing the grammar constant of the  $O(n^3)$ -time algorithm by (Eisner, 1997). Eisner and Satta have developed an  $O(n^7)$ -time parsing algorithm for bilexical tree adjoining grammars, based on the ideas collected in this paper, improving the naive  $O(n^8)$  method. Finally, the theoretical and practical speedup of the given algorithms have been shown. Detailed results of some practical tests can be found in the slides for the talk "Efficient Parsing for Bilexical CF Grammars and Head Automaton Grammars" held by Eisner at ACL99.

## 11. References

- Alshawi H., 1996, Head automata and bilingual tiling: Translation with minimal representations. In *Proceedings of the 34<sup>th</sup> ACL*, pages 167-176, Santa Cruz, CA.
- Charniak E., 1997, Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the 14<sup>th</sup> National Conference on Artificial Intelligence*, pages 598-603, Menlo Park. AAAI Press/MIT Press.
- Collins, M.J., 1997, Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35<sup>th</sup> ACL*, pages 16-23, Madrid, July.
- Mel'čuk I., 1988, *Dependency Syntax: Theory and Practice*. State University of New York Press.
- Eisner J., 1996a, An empirical comparison of probability models for dependency grammar. In *Technical Report IRCS-96-11*, Institute for Research in Cognitive Science, Univ. Of Pennsylvania.
- Eisner J., 1996b, Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th International Conference on Computational Linguistics (COLING-96)*, pages 340-345, Copenhagen.
- Eisner J., 1997, Bilexical grammars and a cubic-time probabilistic parser. In *Proceedings of the 1997 International Workshop on Parsing Technologies*, pages 54-65, MIT, Cambridge, MA.
- Eisner J. and Satta G., 1999, Efficient parsing for bilexical context-free grammars and head-automaton grammars. In *Proceedings of the 37th ACL*, pages 457-464, University of Maryland.
- Eisner J. and Satta G., 2000, A faster parsing algorithm for lexicalized tree adjoining grammars. In *Proceedings of the 5th Workshop on Tree-Adjoining Grammars and Related Formalisms (TAG+5)*, Paris.
- Lafferty J., Sleator D. and Temperley D., 1992, Grammatical trigrams: A probabilistic model of link grammar. In *Proceedings of the AAAI Conf. On Probabilistic Approaches to Natural Language*, October.
- Magerman D., 1995, Statistical decision-tree models for parsing. In *Proceedings of the 33rd ACL*.
- Pollard C. and Sag I., 1994, *Head-Driven Phrase Structure Grammar*. University of Chicago Press.
- Schabes Y., Abeille A., and Joshi A., 1988, Parsing strategies with 'lexicalized' grammars: Application to Tree Adjoining Grammars. In *Proceedings of COLING-88*, Budapest, August.
- Younger D.H., 1967, Recognition and parsing of context-free languages in time  $n^3$ . In *Information and Control*, 10(2):189-208, February.