MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

A.I. Memo No. 812 December, 1984

ON THE COMPLEXITY OF ID/LP PARSING

G. Edward Barton, Jr.

ABSTRACT:

Recent linguistic theories cast surface complexity as the result of interacting subsystems of constraints. For instance, the ID/LP grammar formalism separates constraints on immediate dominance from those on linear order. Shieber (1983) has shown how to carry out *direct parsing* of ID/LP grammars. His algorithm uses ID and LP constraints directly in language processing, without expanding them into a context-free "object grammar." This report examines the computational difficulty of ID/LP parsing. Shieber's purported $O(|G|^2 \cdot n^3)$ runtime bound underestimates the difficulty of ID/LP parsing; the worst-case runtime of his algorithm is exponential in grammar size. A reduction of the vertex-cover problem proves that ID/LP parsing is NP-complete. The growth of internal data structures is the source of difficulty in Shieber's algorithm. The computational and linguistic implications of these results are discussed. Despite the potential for combinatorial explosion, Shieber's algorithm remains better than the alternative of parsing an expanded object grammar.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's artificial intelligence research has been provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-80-C-0505. Support for the author's graduate studies has been provided by the Fannie and John Hertz Foundation. Useful guidance and commentary during the writing of this paper have been provided by Bob Berwick, Michael Sipser, and Joyce Friedman.

1. Introduction

Under most recent linguistic theories, linguistic constraints fall into several subsystems each having its own character. Chomsky (1981:5), for instance, identifies the subtheories of bounding, government, θ-marking, binding, Case, and control, while Shieber (1983:2ff) describes a version of Gazdar and Pullum's GPSG formalism that involves immediate-dominance rules, linear-order constraints, and metarules. When several independent constraints are involved, a rule system that explicitly multiplies out their effects is large, cumbersome, and uninformative.[1] For example, as Shieber (:4) points out, the expanded context-free "object grammar" derived by multiplying out the constraints in a typical GPSG system would contain trillions of rules.

Given the disadvantages of multiplying out the effects of separate systems of constraints, Shieber's (1983) work leads in a welcome direction. Shieber considers how one might do parsing with ID/LP grammars, which involve two orthogonal kinds of rules. ID rules constrain *immediate dominance* irrespective of constituent order ("a sentence can be composed of V with NP and SBAR complements"), while LP rules constrain *linear precedence* among the daughters of any node ("if V and SBAR are sisters, then V must precede SBAR"). Shieber shows how Earley's (1970) algorithm for parsing context-free grammars (CFGs) can be adapted to use the constraints of ID/LP grammars directly, without the combinatorially explosive step of converting the ID/LP grammar into standard context-free form. Instead of multiplying out all of the possible surface interactions among the ID and LP rules, Shieber's algorithm applies them one step at a time as needed. Surely this should work better in a parsing application than applying Earley's algorithm to an expanded grammar with trillions of rules, since the worst-case time complexity of Earley's algorithm is proportional to the square of the grammar size!

Shieber's general approach is on the right track. On pain of having a large and cumbersome rule system, the parser designer should first look to linguistics to find the correct set of constraints on syntactic structure, then discover how to apply some form of those constraints in parsing without multiplying out all possible surface manifestations of their effects.

Nonetheless, nagging doubts about computational complexity remain. Although Shieber (1983:15) claims that his algorithm is identical to Earley's in time complexity, it seems almost too much to hope for that the size of an ID/LP grammar should enter into the time complexity of ID/LP parsing in exactly the same way that the size of a CFG enters into the time complexity of CFG parsing. An ID/LP grammar G can enjoy a huge size advantage over a context-free grammar G' for the same language; for example, if G contains only the rule $S \rightarrow_{ID} abcde$, the corresponding G' contains $5! = 120$ rules. In effect, the claim that Shieber's algorithm has the same time complexity as Earley's algorithm means that this tremendously increased brevity of expression comes free (up to a constant). The paucity of supporting argument in Shieber's article does little to allay these doubts:

> We will not present a rigorous demonstration of time complexity, but it should be clear from the close relation between the presented algorithm and Earley's that the complexity is that of Earley's algorithm. In the

[1] See Barton (1984) for discussion.

> worst case, where the LP rules always specify a unique ordering for the right-hand size of every ID rule, the presented algorithm reduces to Earley's algorithm. Since, given the grammar, checking the LP rules takes constant time, the time complexity of the presented algorithm is identical to Earley's .... That is, it is $O(|G|^2 n^3)$, where $|G|$ is the size of the grammar (number of ID rules) and n is the length of the input. (:14f)

Many questions remain; for example, why should a situation of maximal constraint represent the worst case, as Shieber claims?[2]

The following sections will investigate the complexity of ID/LP parsing in more detail. In brief, the outcome is that Shieber's direct-parsing algorithm usually *does* have a time advantage over the use of Earley's algorithm on the expanded CFG, but that it blows up in the worst case. The claim of $O(|G|^2 n^3)$ time complexity is mistaken; in fact, the worst-case time complexity of ID/LP parsing cannot be bounded by any polynomial in the size of the grammar and input, unless $\mathcal{P} = \mathcal{NP}$. ID/LP parsing is NP-complete.

As it turns out, the complexity of ID/LP parsing has its source in the immediate-domination rules rather than the linear precedence constraints. Consequently, the precedence constraints will be neglected. Attention will be focused on *unordered context-free grammars* (UCFGs), which are exactly like standard context-free grammars except that when a rule is used in a derivation, the symbols on its right-hand side are considered to be unordered and hence may be written in any order. UCFGs represent the special case of ID/LP grammars in which there are no LP constraints. Shieber's ID/LP algorithm can be used to parse UCFGs simply by ignoring all references to LP constraints.

2. Generalizing Earley's algorithm

Shieber generalizes Earley's algorithm by modifying the *progress datum* that tracks progress through a rule. The Earley algorithm uses the position of a dot to track linear advancement through an ordered sequence of constituents. The major predicates and operations on such dotted rules are these:

- A dotted rule is *initialized* with the dot at the left edge, as in $X \rightarrow .ABC$.
- A dotted rule is *advanced* across a terminal or nonterminal that was predicted and has been located in the input by simply moving the dot to the right. For example, $X \rightarrow A.BC$ is advanced across a B by moving the dot to obtain $X \rightarrow AB.C$.
- A dotted rule is *complete* iff the dot is at the right edge. For example, $X \rightarrow ABC.$ is complete.
- A dotted rule *predicts* a terminal or nonterminal iff the dot is immediately before the terminal or nonterminal. For example, $X \rightarrow A.BC$ predicts B.

UCFG rules differ from CFG rules only in that the right-hand sides represent unordered multisets (that is, sets with repeated elements allowed). It is thus appropriate to use successive accumulation of set elements in place of linear advancement through a sequence. In

[2]See section 5; it is in fact the *best* case.

essence, Shieber's algorithm replaces the standard operations on dotted rules with corresponding operations on what will be called dotted UCFG rules:[3]

- A dotted UCFG rule is *initialized* with the empty multiset before the dot and the entire multiset of right-hand elements after the dot, as in $X \rightarrow \{\}.\{A, B, C\}$.
- A dotted UCFG rule is *advanced* across a terminal or nonterminal that was predicted and has been located in the input by simply moving one element from the multiset after the dot to the multiset before the dot. For example, $X \rightarrow \{A\}.\{B, C\}$ is advanced across a B by moving the B to obtain $X \rightarrow \{A, B\}.\{C\}$. Similarly, $X \rightarrow \{A\}.\{B, C, C\}$ may be advanced across a C to obtain $X \rightarrow \{A, C\}.\{B, C\}$.
- A dotted UCFG rule is *complete* iff the multiset after the dot is empty. For example, $X \rightarrow \{A, B, C\}.\{\}$ is complete.
- A dotted UCFG rule *predicts* a terminal or nonterminal iff the terminal or nonterminal is a member of the multiset after the dot. For example, $X \rightarrow \{A\}.\{B, C\}$ predicts B and C.

Given these replacements for operations on dotted rules, Shieber's algorithm operates in the same way as Earley's algorithm. As usual, each state in the parser's state sets consists of a dotted rule tracking progress through a constituent plus the interword position defining the constituent's left edge (Earley, 1970:95, omitting lookahead). The left-edge position is also referred to as the *return pointer* because of its role in the *complete* operation of the parser.

3. The advantages of Shieber's algorithm

The first question to ask is whether Shieber's algorithm saves anything. Is it faster to use Shieber's algorithm on a UCFG than to use Earley's algorithm on the corresponding expanded CFG? Consider the UCFG G_1 that has only the single rule $S \rightarrow abcde$. The corresponding CFG G'_1 has 120 rules spelling out all the permutations of *abcde*: $S \rightarrow abcde$, $S \rightarrow abced$, and so forth. If the string *abcde* is parsed using Shieber's algorithm directly on G_1, the state sets of the parser remain small:[4]

$$
\begin{aligned}
S_0 &: \ [S \rightarrow \{\}.\{a, b, c, d, e\}, 0] \\
S_1 &: \ [S \rightarrow \{a\}.\{b, c, d, e\}, 0] \\
S_2 &: \ [S \rightarrow \{a, b\}.\{c, d, e\}, 0] \\
S_3 &: \ [S \rightarrow \{a, b, c\}.\{d, e\}, 0] \\
S_4 &: \ [S \rightarrow \{a, b, c, d\}.\{e\}, 0] \\
S_5 &: \ [S \rightarrow \{a, b, c, d, e\}.\{\}, 0]
\end{aligned}
$$

In contrast, consider what happens if the same string is parsed using Earley's algorithm on the expanded CFG with its 120 rules. As Figure 1 illustrates, the state sets of the Earley

[3]Shieber's representation differs in some ways from the representation used here, which was developed independently by the author. The differences are generally inessential, but see note 5.

[4]The states related to the auxiliary start symbol and endmarker that are added by some versions of the Earley parser have been omitted for simplicity.

(a) $[S \rightarrow \{a\}.\{b,c,d,e\},0]$

(b)

$[S \rightarrow a.edcb,0]$	$[S \rightarrow a.ecbd,0]$
$[S \rightarrow a.decb,0]$	$[S \rightarrow a.cebd,0]$
$[S \rightarrow a.ecdb,0]$	$[S \rightarrow a.ebcd,0]$
$[S \rightarrow a.cedb,0]$	$[S \rightarrow a.becd,0]$
$[S \rightarrow a.dceb,0]$	$[S \rightarrow a.cbed,0]$
$[S \rightarrow a.cdeb,0]$	$[S \rightarrow a.bced,0]$
$[S \rightarrow a.edbc,0]$	$[S \rightarrow a.dcbe,0]$
$[S \rightarrow a.debc,0]$	$[S \rightarrow a.cdbe,0]$
$[S \rightarrow a.ebdc,0]$	$[S \rightarrow a.dbce,0]$
$[S \rightarrow a.bedc,0]$	$[S \rightarrow a.bdce,0]$
$[S \rightarrow a.dbec,0]$	$[S \rightarrow a.cbde,0]$
$[S \rightarrow a.bdec,0]$	$[S \rightarrow a.bcde,0]$

Figure 1: The use of the Shieber parser on a UCFG can enjoy a large advantage over the use of the Earley parser on the corresponding expanded CFG. After having processed the terminal a while parsing the string *abcde* as discussed in the text, the Shieber parser uses the single state shown in (a) to keep track of the same information for which the Earley parser uses the 24 states in (b).

parser are much larger. In state set S_1, the Earley parser uses $4! = 24$ states to spell out all the possible orders in which the remaining symbols $\{b,c,d,e\}$ could appear. Shieber's modified parser does not spell them out, but uses the single state $[S \rightarrow \{a\}.\{b,c,d,e\},0]$ to summarize them all. Shieber's algorithm should thus be faster, since both parsers work by successively processing all of the states in the state sets.

Similar examples show that the Shieber parser can enjoy an arbitrarily large advantage over the use of the Earley parser on the expanded CFG. Instead of multiplying out all surface appearances ahead of time to produce an expanded CFG, Shieber's algorithm works out the possibilities one step at a time, as needed. This can be an advantage because not all of the possibilities may arise with a particular input.

4. Combinatorial explosion with Shieber's algorithm

The answer to the first question is *yes*, then: it can be more efficient to use Shieber's parser than to use the Earley parser on an expanded "object grammar." The second question to ask is whether Shieber's parser *always* enjoys a large advantage. Does the algorithm blow up in difficult cases?

In the presence of lexical ambiguity, Shieber's algorithm can suffer from combinatorial

explosion. Consider the following UCFG, G_2, in which x is five-ways ambiguous:

$$
\begin{aligned}
S &\rightarrow ABCDE \\
A &\rightarrow a \mid x \\
B &\rightarrow b \mid x \\
C &\rightarrow c \mid x \\
D &\rightarrow d \mid x \\
E &\rightarrow e \mid x
\end{aligned}
$$

What happens if Shieber's algorithm is used to parse the string $xxxxa$ according to this grammar? After the first three occurrences of x have been processed, the state set of Shieber's parser will reflect the possibility that *any three* of the phrases A, B, C, D, and E might have been encountered in the input and *any two* of them might remain to be parsed. There will be $\binom{5}{3} = 10$ states reflecting progress through the rule expanding S, in addition to 5 states reflecting phrase completion and 10 states reflecting phrase prediction (not shown):

$$
\begin{aligned}
S_3: \quad & [S \rightarrow \{A,B,C\}.\{D,E\},0] \quad [S \rightarrow \{A,B,D\}.\{C,E\},0] \\
& [S \rightarrow \{A,C,D\}.\{B,E\},0] \quad [S \rightarrow \{B,C,D\}.\{A,E\},0] \\
& [S \rightarrow \{A,B,E\}.\{C,D\},0] \quad [S \rightarrow \{A,C,E\}.\{B,D\},0] \\
& [S \rightarrow \{B,C,E\}.\{A,D\},0] \quad [S \rightarrow \{A,D,E\}.\{B,C\},0] \\
& [S \rightarrow \{B,D,E\}.\{A,C\},0] \quad [S \rightarrow \{C,D,E\}.\{A,B\},0]
\end{aligned}
$$

In cases like this, Shieber's algorithm enumerates all of the combinations of k elements taken i at a time, where k is the rule length and i is the number of elements already processed. Thus it can be combinatorially explosive.

It is important to note that even in this case, Shieber's algorithm wins out over parsing the expanded CFG with Earley's algorithm. After the same input symbols have been processed, the state set of the Earley parser will reflect the same possibilities as the state set of the Shieber parser: any three of the required phrases might have been located, while any two of them might remain to be parsed. However, the Earley parser has a less concise representation to work with. In place of the state involving $S \rightarrow \{A,B,C\}.\{D,E\}$, for instance, there will be $3! \cdot 2! = 12$ states involving $S \rightarrow ABC.DE$, $S \rightarrow BCA.ED$, and so forth.[5] Instead of a total of 25 states, the Earley state set will contain $135 = 12 \cdot 10 + 15$ states.

In the above case, although the parser could not be sure of the *categorial identities* of the phrases parsed, at least there was no uncertainty about the *number* of phrases and their *extent*. We can make matters even worse for the parser by introducing uncertainty in those areas as well. Let G_3 be the result of replacing every x in G_2 with the empty string ϵ:

$$
\begin{aligned}
S &\rightarrow ABCDE \\
A &\rightarrow a \mid \epsilon \\
B &\rightarrow b \mid \epsilon \\
C &\rightarrow c \mid \epsilon \\
D &\rightarrow d \mid \epsilon \\
E &\rightarrow e \mid \epsilon
\end{aligned}
$$

[5] In contrast to the representation illustrated here, Shieber's representation actually suffers to some extent from the same problem. Shieber (1983:10) uses an ordered sequence instead of a multiset before the dot; consequently, in place of the state involving $S \rightarrow \{A,B,C\}.\{D,E\}$, Shieber would have the $3! = 6$ states involving $S \rightarrow \alpha.\{D,E\}$, where α ranges over the six permutations of ABC.

Then an A, for instance, can be either an a or nothing. Before any input has been read, the first state set S_0 in Shieber's parser must reflect the possibility that the correct parse may include *any of the* $2^5 = 32$ *possible subsets* of A, B, C, D, and E as empty initial constituents. For example, S_0 must include $[S \rightarrow \{A, B, C, D, E\}.\{\}, 0]$ because the input might turn out to be the null string. Similarly, it must include $[S \rightarrow \{A, C, E\}.\{B, D\}, 0]$ because the input might turn out to be *bd* or *db*. Counting all possible subsets in addition to other states having to do with predictions, completions, and the parser's start symbol, there are 44 states in S_0. (There are 338 states in the corresponding state when the expanded CFG G'_3 is used.)

5. The source of the difficulty

Why is Shieber's algorithm potentially exponential in grammar size despite its "close relation" to Earley's algorithm, which has time complexity polynomial in grammar size? The answer lies in the size of the state space that each parser uses. Relative to grammar size, Shieber's algorithm involves a much larger bound than Earley's algorithm on the number of states in a state set. Since the main task of the Earley parser is to perform *scan, predict,* and *complete* operations on the states in each state set (Earley, 1970:97), an explosion in the size of the state sets will be fatal to any small runtime bound.

Given a CFG G_a, how many possible dotted rules are there? Resulting from each rule $X \rightarrow A_1 \ldots A_k$, there are $k + 1$ possible dotted rules. Then the number of possible dotted rules is bounded by $|G_a|$, if this notation is taken to mean the number of symbols that it takes to write G_a down. An Earley state is a pair $[r, i]$, where r is a dotted rule and i is an interword position ranging from 0 to the length n of the input string. Because of these limits, no state set in the Earley parser can contain more than $O(|G_a| \cdot n)$ (distinct) states.

The limited size of a state set allows an $O(|G_a|^2 \cdot n^3)$ bound to be placed on the runtime of the Earley parser. Informally, the argument (due to Earley) runs as follows. The *scan* operation on a state can be done in constant time; the *scan* operations in a state set thus contribute no more than $O(|G_a| \cdot n)$ computational steps. All of the *predict* operations in a state set taken together can add no more states than the number of rules in the grammar, bounded by $|G_a|$, since a nonterminal needs to be expanded only once in a state set regardless of how many times it is predicted; hence the *predict* operations need not take more than $O(|G_a| \cdot n + |G_a|) = O(|G_a| \cdot n)$ steps. Finally, there are the *complete* operations to be considered. A given completion can do no worse than advancing every state in the state set indicated by the return pointer. Therefore, k completions require at most k^2 steps; the *complete* operations in a state set can take no more than $O(|G_a|^2 \cdot n^2)$ steps. Overall, then, it takes no more than $O(|G_a|^2 \cdot n^2)$ steps to process one state set and no more than $O(|G_a|^2 \cdot n^3)$ steps for the Earley parser to process them all.

In Shieber's parser, though, the state sets can grow much larger relative to grammar size. Given a UCFG G_b, how many possible dotted UCFG rules are there? Resulting from a rule $X \rightarrow A_1 \ldots A_k$, there are not $k+1$ possible dotted rules tracking linear advancement, but 2^k possible dotted UCFG rules tracking accumulation of set elements. In the worst case, the grammar contains only one rule and k is on the order of $|G_b|$; hence the number

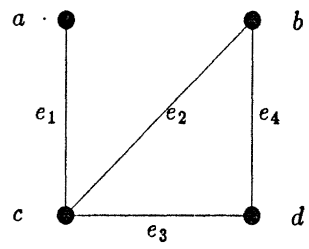


Figure 2: This graph illustrates a trivial instance of the vertex cover problem. The set $\{c, d\}$ is a vertex cover of size 2.

of possible dotted UCFG rules for the whole grammar is not bounded by $|G_b|$, but by $2^{|G_b|}$. (Recall the exponential blowup demonstrated for grammar G_3 in section 4.)

Informally speaking, the reason why Shieber's parser sometimes suffers from combinatorial explosion is that there are exponentially more possible ways to progress through an unordered rule expansion than an ordered one. When disambiguating information is scarce, the parser must keep track of all of them. In the more general task of parsing ID/LP grammars, the most tractable case occurs when constraint from the LP relation is strong enough to force a unique ordering for every rule expansion. Under such conditions, Shieber's parser reduces to Earley's. However, the case of strong constraint represents the *best case* computationally, rather than the *worst case* as Shieber (1983:14) claims.

6. ID/LP parsing is inherently difficult

The worst-case time complexity of Shieber's algorithm is exponential in grammar size rather than quadratic as Shieber (1983:15) believed. Did Shieber simply choose a poor algorithm, or is ID/LP parsing inherently difficult in the general case? In fact, the simpler problem of *recognizing* sentences according to a UCFG is NP-complete.[6] Consequently, unless $\mathcal{P} = \mathcal{NP}$, no algorithm for ID/LP parsing can have a runtime bound that is polynomial in the size of the grammar and input.

The proof of NP-completeness involves reducing the *vertex cover* problem (Garey and Johnson, 1979:46) to the UCFG recognition problem. Through careful construction of the grammar and input string, it is possible to "trick" the parser into solving a known hard problem. The vertex cover problem involves finding a small set of vertices in a graph with the property that every edge of the graph has at least one endpoint in the set. Figure 2 shows a trivial example.

To construct a grammar that encodes the question of whether the graph in Figure 2 has a vertex cover of size 2, first take the vertex names a, b, c, and d as the alphabet. Take

[6]Recognition is simpler than parsing because a recognizer is not required to *recover the structure* of an input string, but only to decide whether the string is *in the language* generated by the grammar: that is, whether or not there *exists* a parse.

$$
\begin{aligned}
START &\rightarrow H_1H_2H_3H_4UUDDDD \\
\\
H_1 &\rightarrow a \mid c \\
H_2 &\rightarrow b \mid c \\
H_3 &\rightarrow c \mid d \\
H_4 &\rightarrow b \mid d \\
\\
U &\rightarrow aaaa \mid bbbb \mid cccc \mid dddd \\
D &\rightarrow a \mid b \mid c \mid d
\end{aligned}
$$

Figure 3: For $k = 2$, the construction described in the text transforms the vertex-cover problem of Figure 2 into this UCFG. A parse exists for the string *aaaabbbbccccdddd* iff the graph in the previous figure has a vertex cover of size ≤ 2.

START as the start symbol. Take H_1 through H_4 as special symbols, one per edge; also take U and D as special dummy symbols.

Next, write the rules corresponding to the edges of the graph. Edge e_1 runs from a to c, so include the rules $H_1 \rightarrow a$ and $H_1 \rightarrow c$. Encode the other edges similarly. Rules expanding the dummy symbols are also needed. Dummy symbol D will be used to soak up excess input symbols, so $D \rightarrow a$ through $D \rightarrow d$ should be rules. Dummy symbol U will also be used to soak up excess input symbols, but U will be allowed to match only when there are four occurrences in a row of the same symbol (one occurrence for each edge). Take $U \rightarrow aaaa$, $U \rightarrow bbbb$, and $U \rightarrow cccc$, and $U \rightarrow dddd$ as the rules expanding U.

Now, what does it take for the graph to have a vertex cover of size $k = 2$? One way to get a vertex cover is to go through the list of edges and underline one endpoint of each edge. If the vertex cover is to be of size 2, the underlining must be done in such a way that only two distinct vertices are ever touched in the process. Alternatively, since there are 4 vertices in all, the vertex cover will be of size 2 if there are $4 - 2 = 2$ vertices left *untouched* in the underlining process. This method of finding a vertex cover can be translated into a UCFG rule as follows:

$$START \rightarrow H_1H_2H_3H_4UUDDDD$$

That is, each H-symbol is supposed to match the name of one of the endpoints of the corresponding edge, in accordance with the rules expanding the H-symbols. Each U-symbol is supposed to correspond to a vertex that was left untouched by the H-matching, and the D-symbols are just there for bookkeeping. Figure 3 lists the complete grammar that encodes the vertex-cover problem of Figure 2.

To make all of this work properly, take

$$\sigma = aaaabbbbccccdddd$$

as the input string to be parsed. (In general, for every vertex name x, include in σ a contiguous run of occurrences of x, one occurrence for each edge in the graph.) The grammar

encodes the underlining procedure by requiring each H-symbol to match one of its endpoints in σ. Since the right-hand side of the *START* rule is unordered, the grammar allows an H-symbol to match anywhere in the input, hence to match any vertex name (subject to interference from other rules that have already matched). Furthermore, since there is one occurrence of each vertex name for every edge, all of the edges could conceivably be matched up with the same vertex; that is, it's impossible to run out of vertex-name occurrences. Consequently, the grammar will allow either endpoint of an edge to be "underlined." The parser will have to figure out which endpoints to choose — in other words, which vertex cover to select. However, the grammar also requires two occurrences of U to match somewhere. U can only match four contiguous identical input symbols that have not been matched in any other way, and thus if the parser chooses a vertex cover that is too large, the U-symbols will not match and the parse will fail. The proper number of D-symbols is given by the length of the input string, minus the number of edges in the graph (to account for the H_i-matches), minus k times the number of edges (to account for the U-matches): in this case, $16 - 4 - (2 \cdot 4) = 4$, as illustrated in the *START* rule.

The net result of this construction is that in order to decide whether σ is in the language generated by the UCFG, the parser must in effect search for a vertex cover of size 2 or less.[7] If a parse exists, an appropriate vertex cover can be read off from beneath the H-symbols in the parse tree; conversely, if an appropriate vertex cover exists, it indicates how to construct a parse. Figure 4 shows the parse tree that encodes a solution to the vertex-cover problem of Figure 2.

The construction shows that vertex-cover problem is reducible to UCFG recognition. Furthermore, the construction of the grammar and input string can be carried out in polynomial time. Consequently, UCFG recognition and the more general task of ID/LP parsing must be computationally difficult. For a more careful and detailed treatment of the reduction and its correctness, see the appendix.

7. Computational implications

The reduction of Vertex Cover shows that the ID/LP parsing problem is NP-complete. Unless $\mathcal{P} = \mathcal{NP}$, the time complexity of ID/LP parsing cannot be bounded by any polynomial in the size of the grammar and input.[8] An immediate conclusion is that complexity analysis must be done carefully: despite its similarity to Earley's algorithm, Shieber's algorithm does not have complexity $O(|G|^2 \cdot n^3)$. For some choices of grammar and input, its internal structures undergo exponential growth. Other consequences also follow.

7.1. Parsing the object grammar

Even in the face of its combinatorially explosive worst-case behavior, Shieber's algo-

[7] If the vertex cover is *smaller* than expected, the D-symbols will soak up the extra contiguous runs that could have been matched by more U-symbols.

[8] Even assuming $\mathcal{P} \neq \mathcal{NP}$, it does not *follow* that the time complexity must be *exponential*, though it seems likely to be. There are functions such as $n^{\log n}$ that fall between polynomials and exponentials. See Hopcroft and Ullman (1979:341).

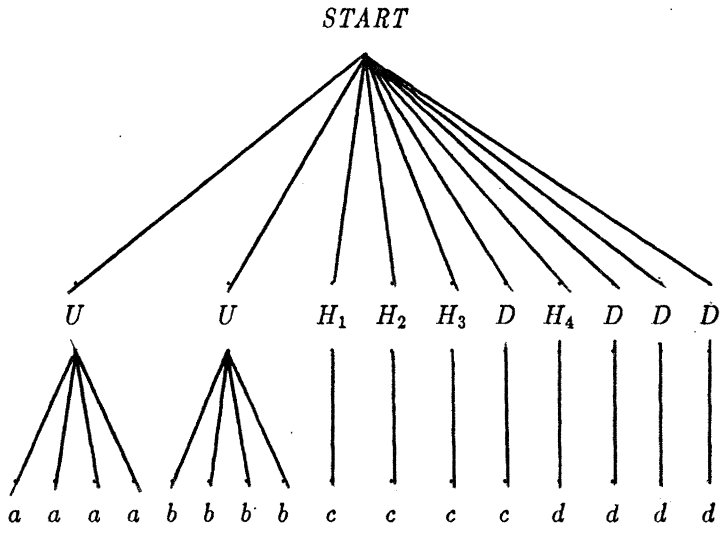


Figure 4: The grammar of Figure 3, which encodes the vertex-cover problem of Figure 2, generates the string $\sigma = aaaabbbbccccdddd$ according to this parse tree. The vertex cover $\{c, d\}$ can be read off from the parse tree as the set of elements dominated by H-symbols.

rithm should not be immediately cast aside. Despite the fact that it sometimes blows up, it still has an advantage over the alternative of parsing the expanded "object grammar." One interpretation of the NP-completeness result is that the general case of ID/LP parsing is inherently difficult; hence it should not be surprising that Shieber's algorithm for solving that problem can sometimes suffer from combinatorial explosion. More significant is the fact that parsing with the expanded CFG blows up in cases that should *not* be difficult. There is nothing inherently difficult about parsing the language that consists of all permutations of the string *abcde*, but while parsing that language the Earley parser can use 24 states or more to encode what the Shieber parser encodes in only one (§3). To put the point another way, the significant fact is not that the Shieber parser can blow up; it is that the use of an expanded CFG blows up *unnecessarily.*

7.2. Is precompilation possible?

The present reduction of Vertex Cover to ID/LP Parsing involves constructing a grammar and input string that *both* depend on the problem to be solved. Consequently, the reduction does not rule out the possibility that through clever programming one might concentrate most of the computational difficulty of ID/LP parsing into a separate *precompilation* stage, dependent on the grammar but independent of the input. According to this optimistic scenario, the entire procedure of preprocessing the grammar and parsing the input string would be as difficult as any NP-complete problem, but after precompilation, the time required for parsing a particular input would be bounded by a polynomial in grammar

size and sentence length.

Regarding the case immediately at hand, Shieber's modified Earley algorithm has no precompilation step.[9] The complexity result implied by the reduction thus applies with full force; any possible precompilation phase has yet to be proposed. Moreover, it is by no means clear that a clever precompilation step is even *possible;* it depends on exactly how $|G|$ and n enter into the complexity function for ID/LP parsing. If n enters as a *factor* multiplying an exponential, precompilation cannot help enough to ensure that the parsing phase will run in polynomial time.

For example, suppose some parsing problem is known to require $2^{|G|} \cdot n^3$ steps for solution.[10] If one is willing to spend, say, $10 \cdot 2^{|G|}$ steps in the precompilation phase, is it possible to reduce parsing-phase complexity to something like $|G|^8 \cdot n^3$? The answer is no. Since by hypothesis it takes at least $2^{|G|} \cdot n^3$ steps to solve the problem, there must be at least $2^{|G|} \cdot n^3 - 10 \cdot 2^{|G|}$ steps left to perform after the precompilation phase. The parameter n is necessarily absent from the precompilation complexity, hence the term $2^{|G|} \cdot n^3$ will eventually dominate.

In a related vein, suppose the precompilation step is conversion from ID/LP to CFG form and the runtime step is the use of the Earley parser on the expanded CFG. Although the precompilation step does a potentially exponential amount of work in producing G' from G, another exponential factor still shows up at runtime because $|G'|$ in the complexity bound $|G'|^2 n^3$ is exponentially larger than the original $|G|$.

7.3. Polynomial-time parsing of a fixed grammar

As noted above, both grammar and input in the current vertex-cover reduction depend on the vertex-cover problem to be solved. The NP-completeness result would be strengthened if there were a reduction that used the same fixed grammar for all vertex-cover problems, for it would then be possible to prove that a precompilation phase would be of little avail. However, unless $\mathcal{P} = \mathcal{NP}$, it is impossible to design such a reduction. Since grammar size is not considered to be a parameter of a fixed-grammar parsing problem, the use of the Earley parser on the object grammar constitutes a polynomial-time algorithm for solving the fixed-grammar ID/LP parsing problem.

Although ID/LP parsing for a fixed grammar can thus be done in cubic time, that fact has little practical significance. The object grammar G' corresponding to a practical ID/LP grammar would be huge, and if $|G'|^2 \cdot n^3$ complexity is too slow, then it remains too slow when $|G'|^2$ is regarded as a constant.

7.4. The power of the UCFG formalism

The Vertex Cover reduction also helps pin down the computational power of the UCFG formalism. As G_1 and G_1' in section 3 illustrated, a UCFG (or an ID/LP grammar) can enjoy

[9]Shieber (1983:15 n. 6) mentions a possible precompilation step, but it is concerned with the LP relation rather than the ID rules.

[10]It is not known whether the worst-case complexity of ID/LP parsing is exponential, since more generally it is not known for sure that $\mathcal{P} \neq \mathcal{NP}$.

considerable brevity of expression compared to the equivalent CFG. The NP-completeness result illuminates this property in two ways. First, the result shows that this brevity of expression is sufficient to allow an instance of any problem in $\mathcal{NP}$ to be stated in a UCFG that is only polynomially larger than the original problem instance. In contrast, if an attempt is made to replicate the current reduction with a CFG rather than UCFG, the necessity of spelling out all the orders in which the H-, U-, and D-symbols might appear makes the CFG more than polynomially larger than the problem instance. Consequently, the reduction fails to establish NP-completeness, which indeed does not hold. Second, the result shows that the increased expressive power does not come free; while the CFG recognition problem can be solved in cubic time or less,[11] unless $\mathcal{P} = \mathcal{NP}$ the general UCFG recognition problem cannot be solved in polynomial time.

The details of the reduction also help pin down how powerful a single UCFG rule can be. If the UCFG formalism is extended to permit ordinary CFG rules in addition to rules with unordered expansions, the grammar that expresses a vertex-cover problem needs only *one* UCFG rule, although that rule may need to be arbitrarily long.

7.5. The role of constraint

Finally, the discussion of section 5 illustrates the way in which the *weakening of constraints* can often make a problem computationally *more difficult*. It might erroneously be thought that weak constraints represent the best case in computational terms, for "weak" constraints sound easy to verify. However, oftentimes the weakening of constraint multiplies the number of possibilities that must be considered in the course of solving a problem. In the case at hand, the removal of constraints on the order in which constituents can appear causes the dependence of parsing complexity on grammar size to grow from $|G|^2$ to $2^{|G|}$.

8. Linguistic implications

Significantly, the key ingredients that can cause difficulties for the ID/LP parsing algorithm are not exotically foreign to linguistic theory. Most current formalisms (*e.g.* GB-theory and GPSG) permit the existence of constituents that are empty on the surface; hence in principle they permit the kind of pathological case illustrated by G_3 in section 4, subject to amelioration by additional constraints. Similarly, a key ingredient of the vertex-cover reduction is lexical ambiguity — acknowledged by every current theory.

Nonetheless, the implications of the NP-completeness result for grammatical theory are fewer than they might seem. The reduction contributes to the necessary goal of understanding the computational power of various mechanisms and formal devices, but it does not (for instance) rule out the use of formalisms that decouple constraints on order from constraints on linear precedence.

Under the assumption that natural languages are efficiently parsable, computational difficulties in parsing a formalism *do* indicate that the formalism itself does not tell the

[11] Since $O(|G|^2 \cdot n^3) < O\big((|G| + n)^3\big)$, the complexity of Earley's algorithm is no worse than cubic in the combined length of grammar and input.

whole story. That is, they point out that the range of possible languages has been incorrectly characterized: the additional constraints that guarantee efficient parsability remain unstated. Since the *general* case of parsing ID/LP grammars is computationally difficult, if the *linguistically relevant* ID/LP grammars are to be efficiently parsable, there must be additional factors that guarantee, say, a certain amount of constraint from the LP relation.[12] (Constraints beyond the bare ID/LP formalism are required on linguistic grounds as well.) Note that the *subset principle* of language acquisition (*cf.* Berwick and Weinberg, 1984:233) would lead the language learner to initially hypothesize strong order constraints, to be weakened only in response to positive evidence.

However, there are other potential ways to guarantee efficient parsability. It might turn out that the principles and parameters of the best grammatical theory permit languages that are not efficiently parsable in the worst case — just as grammatical theory permits sentences that are deeply center-embedded (Miller and Chomsky, 1963).[13] In such a situation, difficult languages or sentences would not be expected to turn up in general use, precisely *because* they would be difficult to process.[14] The factors that guarantee efficient parsability would not be part of grammatical theory because they would result from extragrammatical factors, *i.e.* the resource limitations of the language-processing mechanisms. This "easy way out" is not automatically available, depending as it does on a detailed account of processing mechanisms. For example, in the Earley parser, the difficulty of parsing a construction can vary widely with the amount of lookahead used (if any). Like any other theory, an explanation based on resource limitations must make the right predictions about which constructions will be difficult to parse.

In the same way, the language-acquisition procedure could potentially be the source of some constraints relevant to efficient parsability. Perhaps not all of the languages permitted by the principles and parameters of syntactic theory are *accessible* in the sense that they can potentially be constructed by the language-acquisition component. It is to be expected that language-acquisition mechanisms will be subject to various kinds of limitations just as all other mental mechanisms are. Again, however, concrete conclusions must await a detailed proposal.

---

[12] In the GB-framework of Chomsky (1981), for instance, the syntactic expression of unordered θ-grids at the $\overline{X}$ level is constrained by the principles of Case theory. Endocentricity is another significant constraint. See also Berwick's (1982) discussion of constraints that could be placed on another grammatical formalism — lexical-functional grammar — to avoid a similar intractability result.

[13] Indeed, one may not conclude *a priori* that the languages permitted by linguistic theory are parsable *at all* (Chomsky, 1980).

[14] It is often anecdotally remarked that languages that allow relatively free word order tend to make heavy use of inflections. A rich inflectional system can supply parsing constraints that make up for the lack of ordering constraints; thus the situation we do *not* find is the computationally difficult case of weak constraint.

9. Appendix

This appendix contains the details of a careful reduction of the vertex-cover problem to the UCFG recognition problem. This version of the reduction establishes that the difficulty of UCFG recognition is not due either to the possibility of empty constituents (ϵ-rules) or to the possibility of repeated symbols in rules (*i.e.* to the use of multisets rather than sets). Consequently, it is somewhat different from and more complex than the one sketched in the text.

9.1. Defining unordered context-free grammars

Definition: An *unordered CFG* (UCFG) is a quadruple $\langle N, \Sigma, R, S\rangle$, where:

(a) N is a finite set of *nonterminals.*

(b) Σ disjoint from N is a finite, nonempty set of *terminal symbols.*

(c) R is a nonempty set of *rules* $\langle A, \alpha\rangle$, where $A \in N$ and $\alpha \in (N \cup \Sigma)^*$. The rule $\langle A, \alpha\rangle$ may be written as $A \to \alpha$.

(d) $S \in N$ is the *start symbol.*

Convention: The grammar G and its components N, Σ, R, S need not be explicitly mentioned when clear from context.

Convention: Unless otherwise noted,

(a) $A, A', A_i, \ldots$ denote elements of N;

(b) $a, a', a_i, \ldots$ denote elements of Σ;

(c) $X, Y, X', Y', X_i, Y_i, \ldots$ denote elements of $N \cup \Sigma$;

(d) $\sigma, u, u', u_i, \ldots$ denote elements of Σ^*;

(e) $\alpha, \beta, \gamma, \varphi, \psi$ denote elements of $(N \cup \Sigma)^*$.

Definition: $G = \langle N, \Sigma, R, S\rangle$ is *ϵ-free* iff for every $\langle A, \alpha\rangle \in R$, $|\alpha| \neq 0$.

Definition: $G = \langle N, \Sigma, R, S\rangle$ is *branching* iff for some $\langle A, \alpha\rangle \in R$, $|\alpha| > 1$.

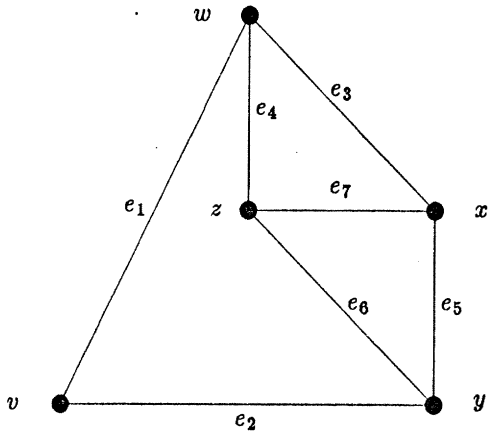
Definition: $G = \langle N, \Sigma, R, S\rangle$ is *duplicate-free* iff for every $\langle A, \alpha\rangle \in R$, $\alpha = Y_1 \ldots Y_n$ and for all $i, j \in [1, n]$, $Y_i = Y_j$ iff $i = j$.

Definition: $G = \langle N, \Sigma, R, S\rangle$ is *simple* iff it is ϵ-free, duplicate-free, and branching.

Note. The notion of a simple UCFG is introduced in order to help pin down the source of any computational difficulties associated with UCFGs. For example, since simple UCFGs are restricted to be duplicate-free, a difficulty that arises with simple UCFGs cannot result from the possibility that a symbol may occur more than once on the right-hand side of a rule.

Definition: $\varphi A \psi \underset{G}{\Rightarrow} \varphi\alpha\psi$ (by r) just in case (for some) $r = \langle A', Y_1 \ldots Y_n\rangle \in R$ and for some permutation ρ of $[1, n]$, $A = A'$ and $\alpha = Y_{\rho(1)} \ldots Y_{\rho(n)}$. If $\varphi \in \Sigma^*$, also write $\varphi A \psi \underset{G}{\Rightarrow}_{\text{lm}} \varphi\alpha\psi$.

Definition: $L(G) = \{\sigma \in \Sigma^* : S \Rightarrow^* \sigma\}$.



$$
\begin{aligned}
V &= \{v, w, x, y, z\} \\
E &= \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\} \\
&\quad \text{with the } e_i \text{ as indicated} \\
k &= 3
\end{aligned}
$$

Figure 5: The triple $\langle V, E, k \rangle$ is an instance of VERTEX COVER. The set $V' = \{v, x, z\}$ is a vertex cover of size $k = 3$.

Definition: An n-step *derivation* of ψ from φ is a sequence $(\varphi_0, \ldots, \varphi_n)$ such that $\varphi_0 = \varphi$, $\varphi_n = \psi$, and for all $i \in [0, n-1]$, $\varphi_i \Rightarrow \varphi_{i+1}$. If it is also true for all i that $\varphi_i \Rightarrow_{\text{lm}} \varphi_{i+1}$, say that the derivation is *leftmost.*

9.2. Defining the computational problems

Definition: A possible instance of the problem VERTEX COVER is a triple $\langle V, E, k \rangle$, where $\langle V, E \rangle$ is a finite graph with at least one edge and at least two vertices, $k \in \mathbf{N}$, and $k < |V|$.[15] VERTEX COVER itself consists of all possible instances $\langle V, E, k \rangle$ such that for some $V' \subseteq V$, $|V'| \leq k$ and for all edges $e \in E$, at least one endpoint of e is in V'. (Figure 5 gives an example of a VERTEX COVER instance.)

Fact: VERTEX COVER is NP-complete. (Garey and Johnson, 1979:46)

Definition: A possible instance of the problem SIMPLE UCFG RECOGNITION is a pair $\langle G, \sigma \rangle$, where G is a simple UCFG and $\sigma \in \Sigma^*$. SIMPLE UCFG RECOGNITION itself consists of all possible instances $\langle G, \sigma \rangle$ such that $\sigma \in L(G)$.

Notation: Take $\|\cdot\|$ to be any reasonable measure of the encoded input length for a computational problem; continue to use $|\cdot|$ for set cardinality and string length. It is reasonable to require that if S is a set, $k \in \mathbf{N}$, and $|S| > k$, then $\|S\| > \|k\|$; that is, the encoding of

[15] This formulation differs trivially from the one cited by Garey and Johnson.

numbers is better than unary. It is also reasonable to require that $\|\langle \ldots, x, \ldots \rangle\| \geq \|x\|$.

9.3. The UCFG recognition problem is in NP

Lemma 9.1: Let $(\varphi_0, \ldots, \varphi_k)$ be a shortest leftmost derivation of φ_k from φ_0 in a branching ϵ-free UCFG. If $k > |N| + 1$ then $|\varphi_k| > |\varphi_0|$.

Proof. There exists some sequence of rules $\langle A_0, \alpha_0 \rangle, \ldots, \langle A_{k-1}, \alpha_{k-1} \rangle$ such that for all $i \in [0, k-1]$, $\varphi_i \Rightarrow_{\text{lm}} \varphi_{i+1}$ by $\langle A_i, \alpha_i \rangle$. Since G is ϵ-free, $|\varphi_{i+1}| \geq |\varphi_i|$ always.

Case 1. For some i, $|\alpha_i| > 1$. Then $|\varphi_{i+1}| > |\varphi_i|$. Hence $|\varphi_k| > |\varphi_0|$.

Case 2. For every i, $|\alpha_i| = 1$. Then there exist u, γ such that for every $i \in [0, k-2]$, there is $A'_i \in N$ such that $\varphi_{i+1} = uA'_i\gamma$. Suppose the A'_i are all distinct. Then $|N| \geq k - 1$, hence $|N| + 1 \geq k$, hence $|N| + 1 > |N| + 1$, which is impossible. Hence for some $i, j \in [0, k-2]$, $i < j$, $A'_i = A'_j$. Hence $\varphi_{i+1} = \varphi_{j+1}$, since $[1, 1]$ has only one permutation. Then $(\varphi_0, \ldots, \varphi_i, \varphi_{j+1}, \ldots, \varphi_k)$ is a leftmost derivation of φ_k from φ_0 and has length less than k, which is also impossible.

Then $|\varphi_k| > |\varphi_0|$. □

Corollary 9.2: If G is a branching ϵ-free UCFG and $\sigma \in L(G)$ then σ has a leftmost derivation of length at most $|\sigma| \cdot m$, where $m = |N| + 2$.

Proof. Let $(\varphi_0, \ldots, \varphi_k)$ be a shortest leftmost derivation of σ from S. Suppose $k > |\sigma| \cdot m$. Consider the sub-derivations

$$\begin{array}{l} (\varphi_0, \ldots, \varphi_m) \\ (\varphi_m, \ldots, \varphi_{2m}) \\ \quad \vdots \\ (\varphi_{(|\sigma|-1)\cdot m}, \ldots, \varphi_{|\sigma|\cdot m}) \\ (\varphi_{|\sigma|\cdot m}, \ldots, \varphi_k). \end{array}$$

Each one except the last has m steps and $m > |N| + 1$. Then by lemma,

$$|\varphi_{|\sigma|\cdot m}| > |\varphi_{(|\sigma|-1)\cdot m}| > \cdots > |\varphi_m| > |\varphi_0| = 1.$$

Then $|\sigma| \geq 1 + |\sigma|$, which is impossible. Hence $k \leq |\sigma| \cdot m$. □

Lemma 9.3: Π = SIMPLE UCFG RECOGNITION is in the computational class $\mathcal{NP}$.

Proof. Let $G = \langle N, \Sigma, R, S \rangle$ be a simple UCFG and $\sigma \in \Sigma^*$. Consider the following nondeterministic algorithm with input $\langle G, \sigma \rangle$:

Step 1. Write down $\varphi_0 = S$.

Step 2. Perform the following steps for i from 0 to $|\sigma| \cdot m - 1$, where $m = |N| + 2$.

(a) Express φ_i as $u_i A_i \gamma_i$ by finding the leftmost nonterminal, or loop if impossible.

(b) Guess a rule $\langle A_i, Y_{i,1} \ldots Y_{i,k_i} \rangle \in R$ and a permutation ρ_i of $[1, k_i]$, or loop if there is no such rule.

(c) Write down $\varphi_{i+1} = u_i Y_{i,\rho_i(1)} \dots Y_{i,\rho_i(k_i)} \gamma_i$.

(d) If $\varphi_{i+1} = \sigma$ then halt.

Step 3. Loop.

It should be apparent that the algorithm runs in time at worst polynomial in $\|\langle G, \sigma \rangle\|$; note that the length of φ_i increases by at most a constant amount on each iteration.

Assume $\langle G, \sigma \rangle \in \Pi$. Then σ has a leftmost derivation of length at most $|\sigma| \cdot m$ by Corollary 9.2; hence the nondeterministic algorithm will be able to guess it and will halt. Conversely, suppose the algorithm halts on input $\langle G, \sigma \rangle$. On the iteration when the algorithm halts, the sequence $(\varphi_0, \dots, \varphi_{i+1})$ will constitute a leftmost derivation of σ from S; hence $\sigma \in L(G)$ and $\langle G, \sigma \rangle \in \Pi$.

Then there is a nondeterministic algorithm that runs in polynomial time and accepts exactly Π. Hence $\Pi \in \mathcal{NP}$. □

9.4. The UCFG recognition problem is NP-complete

Lemma 9.4: Let $\langle V, E, k \rangle = \langle V, \{e_i\}, k \rangle$ be a possible instance of VERTEX COVER. Then it is possible to construct, in time polynomial in $\|V\|$, $\|E\|$, and k, a simple UCFG $G(V, E, k)$ and a string $\sigma(V, E, k)$ such that

$$\begin{aligned} & \langle G(V, E, k), \sigma(V, E, k) \rangle \in \text{SIMPLE UCFG RECOGNITION} \\ \text{iff} \quad & \langle V, E, k \rangle \in \text{VERTEX COVER}. \end{aligned}$$

Proof. Construct $G(V, E, k)$ as follows. Let the set N of nonterminals consist of the following symbols not in V:

$$\begin{aligned} & START, U, D, \\ & H_i \text{ for } i \in [1, |E|], \\ & U_i \text{ for } i \in [1, |V| - k], \\ & D_i \text{ for } i \in [1, |E| \cdot (k-1)]. \end{aligned}$$

$\|N\|$ will be at worst polynomial in $\|E\|$, $\|V\|$, and k for a reasonable length measure. Define the terminal vocabulary Σ to consist of subscripted symbols as follows:

$$\Sigma = \{a_i : a \in V, i \in [1, |E|]\}.$$

Designate $START$ as the start symbol. Include the following as members of the rule set R:

(a) Include the rule

$$START \to H_1 \dots H_{|E|} U_1 \dots U_{|V|-k} D_1 \dots D_{|E| \cdot (k-1)}.$$

(b) For each $e_i \in E$, include the rules

$$\{H_i \to a_i : a \text{ an endpoint of } e_i\}.$$

$$START \rightarrow H_1H_2H_3H_4H_5H_6H_7U_1U_2D_1D_2D_3D_4D_5D_6D_7D_8D_9D_{10}D_{11}D_{12}D_{13}D_{14}$$

$$\begin{array}{lll} H_1 \rightarrow v_1 \mid w_1 & H_2 \rightarrow v_2 \mid y_2 & H_3 \rightarrow w_3 \mid x_3 \\ H_4 \rightarrow w_4 \mid z_4 & H_5 \rightarrow x_5 \mid y_5 & H_6 \rightarrow y_6 \mid z_6 \\ H_7 \rightarrow x_7 \mid z_7 & & \end{array}$$

$$\begin{array}{lll} U_1 \rightarrow U & U_2 \rightarrow U & U_3 \rightarrow U \\ U_4 \rightarrow U & & \end{array}$$

$$\begin{array}{rl} U \rightarrow & v_1v_2v_3v_4v_5v_6v_7 \mid w_1w_2w_3w_4w_5w_6w_7 \mid x_1x_2x_3x_4x_5x_6x_7 \\ \mid & y_1y_2y_3y_4y_5y_6y_7 \mid z_1z_2z_3z_4z_5z_6z_7 \end{array}$$

$$\begin{array}{lll} D_1 \rightarrow D & D_2 \rightarrow D & D_3 \rightarrow D \\ D_4 \rightarrow D & D_5 \rightarrow D & D_6 \rightarrow D \\ D_7 \rightarrow D & D_8 \rightarrow D & D_9 \rightarrow D \\ D_{10} \rightarrow D & D_{11} \rightarrow D & D_{12} \rightarrow D \\ D_{13} \rightarrow D & D_{14} \rightarrow D & \end{array}$$

$$\begin{array}{rl} D \rightarrow & v_1 \mid v_2 \mid v_3 \mid v_4 \mid v_5 \mid v_6 \mid v_7 \mid w_1 \mid w_2 \mid w_3 \mid w_4 \mid w_5 \mid w_6 \mid w_7 \\ \mid & x_1 \mid x_2 \mid x_3 \mid x_4 \mid x_5 \mid x_6 \mid x_7 \mid y_1 \mid y_2 \mid y_3 \mid y_4 \mid y_5 \mid y_6 \mid y_7 \\ \mid & z_1 \mid z_2 \mid z_3 \mid z_4 \mid z_5 \mid z_6 \mid z_7 \end{array}$$

Figure 6: The construction of Lemma 9.4 produces this grammar when applied to the VERTEX COVER problem of Figure 5. The H-symbols ensure that the solution that is found must hit each of the edges, while the U-symbols ensure that enough elements of V remain untouched to satisfy the requirement $|V'| \leq k$. The D-symbols are dummies that absorb excess input symbols. A shorter grammar than this will suffice if the grammar is not required to be duplicate-free.

(c) For each $i \in [1, |V| - k]$, include the rule $U_i \rightarrow U$. Also include the rules

$$\{U \rightarrow a_1 \ldots a_{|E|} : a \in V\}.$$

(d) For each $i \in [1, |E| \cdot (k-1)]$, include the rule $D_i \rightarrow D$. Also include the rules

$$\{D \rightarrow a : a \in \Sigma\}.$$

Take $G(V, E, k)$ to be $\langle N, \Sigma, R, START \rangle$. (Figure 6 shows the result of applying this construction to the VERTEX COVER instance of Figure 5.)

Let $h : [1, |V|] \to V$ be some standard enumeration of the elements of V. Construct $\sigma(V, E, k)$ as $h(1)_1 \ldots h(1)_{|E|} \ldots h(|V|)_1 \ldots h(|V|)_{|E|}$; thus $\sigma(V, E, k)$ will have length $|E| \cdot |V|$.

It is easy to see that $\|\langle G(V, E, k), \sigma(V, E, k)\rangle\|$ will be at worst polynomial in $\|E\|$, $\|V\|$, and k for reasonable $\|\cdot\|$. It will also be possible to *construct* the grammar and string in polynomial time. Finally, note that given the definition of a possible instance of VERTEX COVER, the grammar will be branching, ϵ-free, and duplicate-free, hence simple.

Now suppose $\langle V, E, k\rangle \in$ VERTEX COVER. Then there exist $V' \subseteq V$ and $f : E \to V'$ such that $|V'| \leq k$ and for every $e \in E$, $f(e)$ is an endpoint of e. E is nonempty by hypothesis and V' must hit every edge, hence $|V'|$ cannot be zero. Construct a parse tree for $\sigma(V, E, k)$ according to $G(V, E, k)$ as follows.

Step 1. Number the elements of $V - V'$ as $\{x_i : i \in [1, |V - V'|]\}$. For each x_i where $i \leq |V| - k$, construct a node dominating the substring $(x_i)_1 \ldots (x_i)_{|E|}$ of $\sigma(V, E, k)$ and label it U. Then construct a node dominating only the U-node and label it U_i. Note that the available symbols U_i are numbered from 1 to $|V| - k$, so it is impossible to run out of U-symbols. Also, $|V'| \leq k$ and $V' \subseteq V$, hence $|V - V'| \leq |V| - k$, so all of the U-symbols will be used. Finally, note that $U \to a_1 \ldots a_{|E|}$ is a rule for any $a \in S$ and that $U_i \to U$ is a rule for any U_i.

Step 2. For each $e_i \in E$, construct a node dominating the (unique) occurrence of $f(e_i)_i$ in $\sigma(V, E, k)$ and label it H_i. Step 2 cannot conflict with step 1 because $f(e_i) \in V'$, hence $f(e_i) \notin V - V'$. Different parts of step 2 cannot conflict with each other because each one affects a symbol with a different subscript. Also note that $f(e_i)$ is an endpoint of e_i and that $H_i \to a_i$ is a rule for any $e_i \in E$ and a an endpoint of e_i.

Step 3. Number all occurrences of terminals in $\sigma(V, E, k)$ that were not attached in step 1 or step 2. For the ith such occurrence, construct a node dominating the occurrence and label it D. Then construct another node dominating the D-node and label it D_i. Note that the stock of D-symbols runs from 1 to $(k-1) \cdot |E|$. Exactly $(|V| - k) \cdot |E|$ symbols of $\sigma(V, E, k)$ were accounted for in step 1. Also, exactly $|E|$ symbols were accounted for in step 2. The length of $\sigma(V, E, k)$ is $|V| \cdot |E|$, hence exactly

$$\begin{aligned} |V| \cdot |E| - (|V| - k) \cdot |E|) - |E| &= |V| \cdot |E| - |V| \cdot |E| + k \cdot |E| - |E| \\ &= (k-1) \cdot |E| \end{aligned}$$

symbols remain at the beginning of step 3. $D \to a$ is a rule for any $a \in \Sigma$; $D_i \to D$ is a rule for any D_i.

Step 4. Finally, construct a node labeled *START* that dominates all of the H_i, U_i, and D_i nodes constructed in steps 1, 2, and 3. The rule

$$START \to H_1 \ldots H_{|E|} U_1 \ldots U_{|V|-k} D_1 \ldots D_{|E| \cdot (k-1)}$$

is in the grammar. Note also that nodes labeled $H_1, \ldots, H_{|E|}$ were constructed in step 2, nodes labeled $U_1, \ldots, U_{|V|-k}$ were constructed in step 1, and nodes labeled $D_1, \ldots, D_{|E| \cdot (k-1)}$ were constructed in step 3. Hence the application of the rule is in accord with the grammar.

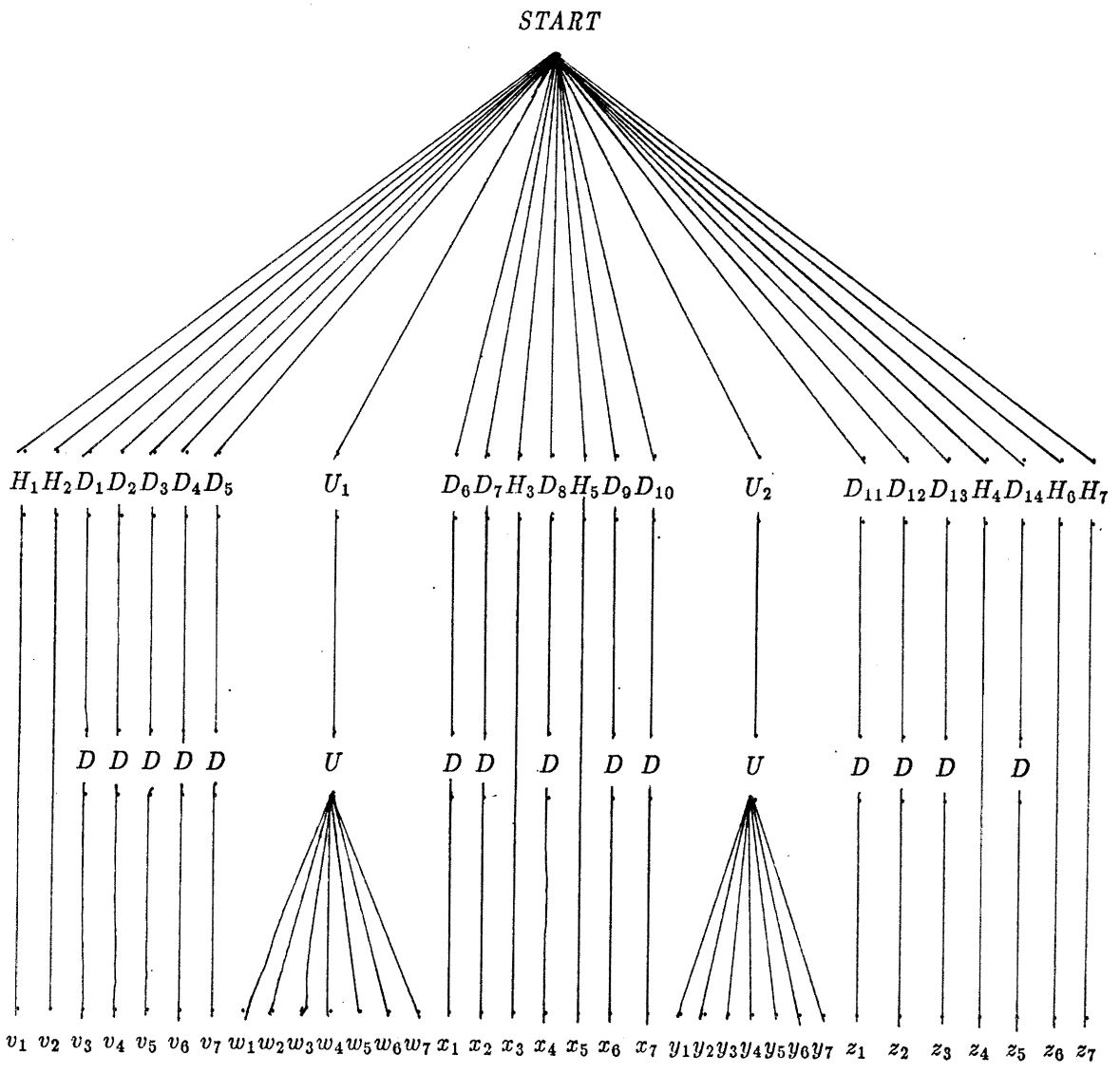


Figure 7: This parse tree shows how the grammar shown in Figure 6 can generate the string $\sigma(V, E, k)$ constructed in Lemma 9.4 for the VERTEX COVER problem of Figure 5. The corresponding VERTEX COVER solution $V' = \{v, x, z\}$ and its intersection with the edges can be read off by noticing which terminals the H-symbols dominate.

Then $\sigma(V, E, k) \in L(G)$. (Figure 7 illustrates the application of this parse-tree construction procedure to the grammar and input string derived from the VERTEX COVER example in Figure 5.)

Conversely, suppose $\sigma(V, E, k) \in L(G)$. Then the derivation of $\sigma(V, E, k)$ from *START* must begin with the application of the rule

$$START \rightarrow H_1 \ldots H_{|E|} U_1 \ldots U_{|V|-k} D_1 \ldots D_{|E|\cdot(k-1)}$$

and each H_i must later be expanded as some subscripted terminal $g(H_i)$. Define $f(e_i)$ to be $g(H_i)$ without the subscript; then by construction of the grammar, $f(e_i)$ is an endpoint of e_i for all $e_i \in E$. Define $V' = \{f(e_i) : e_i \in E\}$; then it is apparent that $V' \subseteq V$ and that V' contains at least one endpoint of e_i for all $e_i \in E$. Also, each U_i for $i \in [1, |V| - k]$ must be expanded as U, then as some substring $(a_i)_1 \ldots (a_i)_{|E|}$ of $\sigma(V, E, k)$.[16] Since the substrings dominated by the H_i and U_i must all be disjoint, and since there are only $|E|$ subscripted occurrences of any single symbol from V in $\sigma(V, E, k)$, there must be $|V| - k$ distinct elements of V that are not dominated in any of their subscripted versions by any H_i. Then $|V - V'| \geq |V| - k$. Since in addition $V \subseteq V'$, $|V'| \leq k$. Then $\langle V, E, k \rangle \in$ VERTEX COVER. □

Theorem 1: SIMPLE UCFG RECOGNITION is NP-complete.

Proof. SIMPLE UCFG RECOGNITION is in the class $\mathcal{NP}$ by Lemma 9.3, hence a polynomial-time reduction of VERTEX COVER to SIMPLE UCFG RECOGNITION is sufficient. Let $\langle V, E, k \rangle$ be a possible instance of VERTEX COVER. Let G be $G(V, E, k)$ and σ be $\sigma(V, E, k)$ as constructed in Lemma 9.4. Note that G is simple.

The construction of G and σ can, by lemma, be carried out at time at worst polynomial in $\|E\|$, $\|V\|$, and k. Also by lemma, $\langle G, \sigma \rangle \in$ SIMPLE UCFG RECOGNITION iff $\langle V, E, k \rangle \in$ VERTEX COVER. k is not polynomial in $\|k\|$ under a reasonable encoding scheme. However, $|E| > k$, hence $\|E\| \geq \|k\|$; also $\|\langle V, E, k \rangle\| \geq \|E\|$, hence $\|\langle V, E, k \rangle\| \geq k$, all by properties assumed to hold of $\|\cdot\|$. Then G and σ can in fact be constructed in time at worst polynomial in $\|\langle V, E, k \rangle\|$.

Hence the VERTEX COVER problem is polynomial-time reduced to SIMPLE UCFG RECOGNITION. □

[16] The grammar would allow the substring $(a_i)_1 \ldots (a_i)_{|E|}$ to appear in any permutation, but in $\sigma(V, E, k)$ it appears only in the indicated order.

10. References

Barton, E. (1984). "Toward a Principle-Based Parser," A.I. Memo No. 788, M.I.T. Artificial Intelligence Laboratory, Cambridge, Mass.

Berwick, R. (1982). "Computational Complexity and Lexical-Functional Grammar," *American Journal of Computational Linguistics* 8.3-4:97–109.

Berwick, R., and A. Weinberg (1984). *The Grammatical Basis of Linguistic Performance.* Cambridge, Mass.: M.I.T. Press.

Chomsky, N. (1980). *Rules and Representations.* New York: Columbia University Press.

Chomsky, N. (1981). *Lectures on Government and Binding.* Dordrecht, Holland: Foris Publications.

Earley, J. (1970). "An Efficient Context-Free Parsing Algorithm," *Comm. ACM* 13.2:94–102.

Garey, M., and D. Johnson (1979). *Computers and Intractability.* San Francisco: W. H. Freeman and Co.

Hopcroft, J., and J. Ullman (1979). *Introduction to Automata Theory, Languages, and Computation.* Reading, Massachusetts: Addison-Wesley.

Miller, G., and N. Chomsky (1963). "Finitary Models of Language Users," in R. D. Luce, R. R. Bush, and E. Galanter, eds., *Handbook of Mathematical Psychology,* vol. II, 419–492. New York: John Wiley and Sons, Inc.

Shieber, S. (1983). "Direct Parsing of ID/LP Grammars." Technical Report 291R, SRI International, Menlo Park, California. Also appears in *Linguistics and Philosophy* 7:2.