Fast Context-Free Grammar Parsing Requires Fast Boolean Matrix Multiplication

LILLIAN LEE

Cornell University, Ithaca, New York

Abstract. In 1975, Valiant showed that Boolean matrix multiplication can be used for parsing context-free grammars (CFGs), yielding the asympotically fastest (although not practical) CFG parsing algorithm known. We prove a dual result: any CFG parser with time complexity $O(gn^{3-\epsilon})$, where g is the size of the grammar and n is the length of the input string, can be efficiently converted into an algorithm to multiply $m \times m$ Boolean matrices in time $O(m^{3-\epsilon/3})$. Given that practical, substantially subcubic Boolean matrix multiplication algorithms have been quite difficult to find, we thus explain why there has been little progress in developing practical, substantially subcubic general CFG parsers. In proving this result, we also develop a formalization of the notion of parsing.

Categories and Subject Descriptors: F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems; I.2.7 [**Artificial Intelligence**]: Natural Language Processing—*language parsing and understanding*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Boolean matrix multiplication, context-free grammar parsing

1. *Introduction*

The context-free grammar (CFG) formalism, introduced by Chomsky [1956], has enjoyed wide use in a variety of fields. CFGs have been used to model the structure of programming languages, human languages, and even biological data such as the sequences of nucleotides making up DNA and RNA [Aho et al. 1986; Jurafsky and Martin 2000; Durbin et al. 1998].

A preliminary conference version of this article appeared in the *Proceedings of the 35th Annual Meeting of the Association for Computational Lingusitics*, pp. 9–15 (thanks to those reviewers for their comments and suggestions).

This material is based upon work supported in part by the National Science Foundation (NSF) under grant number IRI-9350192, an NSF Graduate Fellowship, and an AT&T GRPW/ALFP grant. Any opinions, findings, and conclusions or recommendations expressed in this article are those of the author and do not necessarily reflect the views of the National Science Foundation.

Author's address: Computer Science Department, Cornell University, 4130 Upson Hall, Ithaca, NY 14853, e-mail: llee@cs.cornell.edu.

CFGs are generative systems, where strings are derived via successive applications of rewriting rules. In practice, however, the goal generally is not to generate valid strings from a grammar. Rather, one typically already has some string of interest, such as a C program or an English sentence, in hand, and the goal is to analyze—*parse*—the string with respect to the grammar.

Canonical methods for general CFG parsing are the CKY algorithm [Kasami 1965; Younger 1967] and Earley's algorithm [Earley 1970]. Both have a worst-case running time of $O(gn^3)$ for a CFG of size g and string of length n [Graham et al. 1980], although CKY requires the input grammar to be in Chomsky normal form in order to achieve this time bound. Unfortunately, cubic dependence on the string length is prohibitively expensive in applications such as speech recognition, where responses must be made in real time, or in situations where the input sequences are very long, as in computational biology.

Asymptotically faster parsing algorithms do exist. Graham et al. [1980] give a variant of Earley's algorithm that is based on the so-called "four Russians" algorithm [Arlazarov et al. 1970] for Boolean matrix multiplication (BMM); it runs in time $O(gn^3/\log n)$. Rytter [1985] further modifies this parser by a compression technique, improving the dependence on the string length to $O(n^3/\log^2 n)$. But Valiant's [1975] parsing method, which reorganizes the computations of CKY, is the asymptotically fastest known. It also uses BMM; its worst-case running time for a grammar in Chomsky normal form is proportional to $M(n)$, where $M(m)$ is the time it takes to multiply two $m \times m$ Boolean matrices together.

Since these subcubic parsing algorithms all depend on Boolean matrix multiplication, it is natural to ask how fast BMM can be performed in practice. The asymptotically fastest way known to perform BMM is to rely on algorithms for multiplying arbitrary matrices. There exist matrix multiplication algorithms with time complexity $O(m^{3-\delta})$, thus improving over the standard algorithm's $O(m^3)$ running time; for instance, Strassen's [1969] has a worst-case running time of $O(m^{2.81})$, and the fastest currently known, due to Coppersmith and Winograd [1987; 1990], has time complexity $O(m^{2.376})$. (See Strassen [1990] for a historical account, plotted graphically in Figure 1.) Unfortunately, the constants involved in the subcubic algorithms improving on Strassen's result are so large that these *fast* algorithms cannot be used in practice. As for Strassen's method itself, its practicality is ambiguous: empirical studies show that the "cross-over" point—the matrix size at which it becomes better to use Strassen's method—is above 100 [Bailey 1988; Thottethodi et al. 1998]. In summary, despite decades of research effort, there has been little success at finding a clearly practical, simple, fast matrix multiplication algorithm.

One might therefore hope to find a way to speed up CFG parsing without relying on matrix multiplication. However, the main theorem of this article is that fast CFG parsing *requires* fast Boolean matrix multiplication, in the following precise sense: any parser running in time $O(gn^{3-\epsilon})$ that represents parse data in a retrieval-efficient way can be converted with little computational overhead into an $O(m^{3-\epsilon/3})$ BMM algorithm.

The restriction of our result to parsers with a linear dependence on the grammar size is crucial for relating subcubic parsing to subcubic BMM. However, as discussed in Section 2.3, this restriction is a reasonable one since canonical parsing algorithms such as CKY and Earley's algorithm have this property, and furthermore, in domains like natural language processing, the grammar size is often the dominating factor.

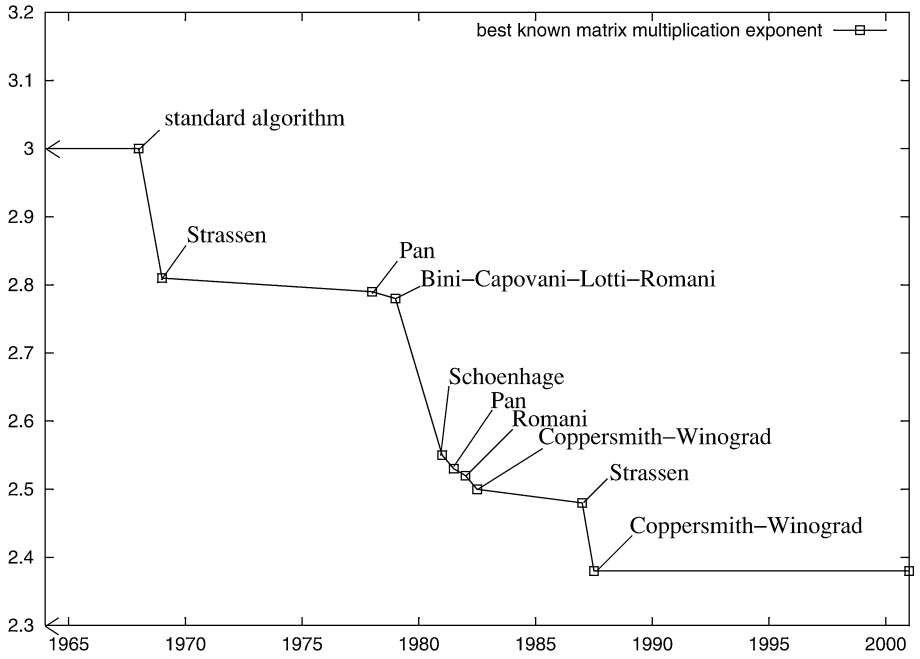

FIG. 1. Lowest known upper bound on the exponent ω for the complexity of matrix multiplication. For instance, before 1969, the fastest known algorithm for matrix multiplication took proportional to m^3 steps ($\omega = 3$).

Our theorem, together with the fact that it has been quite difficult to find practical fast matrix multiplication algorithms, explains why there has been little success to date in developing practical CFG parsers running in substantially subcubic time.

2. *The Parsing Problem: A Formalization*

In this section, we motivate and set forth a formalization of the parsing problem.

2.1. MOTIVATION FOR OUR DEFINITION. In formal language theory, emphasis has been placed on the *recognition* or membership problem: deciding whether or not a given string can be derived by a grammar. However, we concentrate here on the *parsing* problem: finding the parse structure, or analysis, assigned to a string by a grammar. (In the case of *ambiguous* strings, multiple parses exist; we address this point below.)

From a theoretical standpoint, the two problems are almost equivalent. Recognition obviously reduces to parsing, and indeed to our knowledge there are no CFG recognition algorithms that do not implicitly compute parse information. Conversely, Ruzzo [1979] demonstrated that any CFG recognition algorithm that is not already an implicit parser can be converted into an algorithm that returns a (single) parse of the input string w, at a cost of only a factor of $O(\log |w|)$ slowdown.

In practice, however, the parsing problem is much more compelling than the membership problem. Understanding the structure of the input string is crucial to

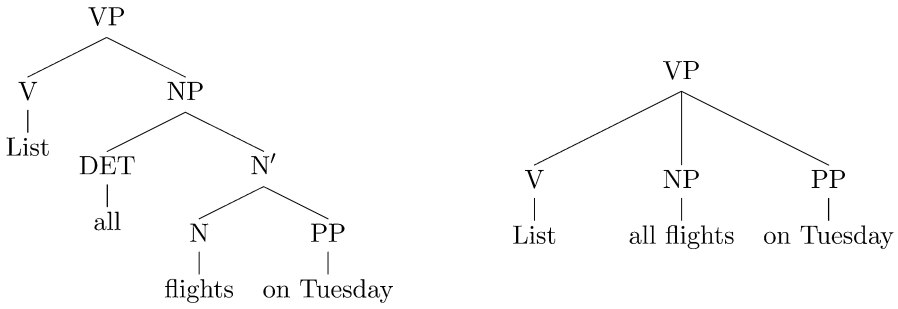

FIG. 2. Two different parse trees for the sentence "List all flights on Tuesday." The labels on the interior nodes denote linguistic categories.

programming language compilation, natural language understanding, RNA shape determination, and so on. In fact, in speech recognition systems, a useful assumption is that any input utterance is somehow "valid," even if it is ungrammatical, thus making the recognition problem trivial. However, different parses of the input sentence may lead to radically different interpretations. For example, the classic sentence "List all flights on Tuesday" has two different parses (see Figure 2): one indicates that all flights taking off on Tuesday should be listed right now, whereas the other asks to wait until Tuesday, and then list all flights regardless of their departure date. Another well-known ambiguous sentence is "I saw the man with the telescope"; observe that here the two possible interpretations seem to be about equally likely.

The fact that some input strings are ambiguous raises the question of what we should require the output of a parsing algorithm to be: any *single* parse of the input string (Ruzzo's reduction of parsing to recognition uses this model), or *all possible* parses? In practice, since multiple analyses may be valid (as in the natural language examples above), it is clear that any practical parser should return all parses.

It remains to determine what the format of the output parses should be. One problem is that there exist grammars for which the number of parse trees for strings of length n grows exponentially in n; for example, consider the Chomsky normal form CFG with productions $S \rightarrow SS|a$.[1] Hence, a compressed representation of the parse structures must be used; otherwise, every parser could take exponential time just to print its output. However, we must be careful to impose restrictions on the compression rate: after all, we could perversely consider the input string itself to be a (rather inconvenient) representation of all its parse trees [Ruzzo 1979]. We thus require practical parsers to output all the parses of an input string in a representation that is both compact and yet allows efficient retrieval of parse information. In the next subsection, we make this notion precise.

2.2. C-PARSING OF CONTEXT-FREE GRAMMARS. We use the usual definition of a context-free grammar (CFG) as a 4-tuple $G = (\Sigma, V, R, S)$, where Σ is the set of terminals, V is the set of nonterminals, R is the set of rewrite rules or productions, and $S \in V$ is the start symbol. Given a string $w = w_1 w_2 \cdots w_n$ in Σ^*,

[1] If we do not impose any restrictions on the form of the grammar, then an *infinite* number of parse trees can be produced for a single string; for example, consider the production set $S \rightarrow S|a$.

where each w_i is an element of Σ, we use the notation w_i^j to denote the substring $w_i w_{i+1} \cdots w_{j-1} w_j$. The *size* of G, denoted by $|G|$, is the sum of the lengths of all productions in R.

Our notion of necessary parse information is based on the concept of CFG *c-derivations*, which are substring derivations that are consistent with some parse of the entire input string.

Definition 1. Let $G = (\Sigma, V, R, S)$ be a *CFG*, and let $w = w_1 w_2 \cdots w_n$, $w_i \in \Sigma$. A nonterminal $A \in V$ *c-derives* (consistently derives) w_i^j if and only if the following conditions hold:

- $A \Rightarrow^* w_i^j$, and
- $S \Rightarrow^* w_1^{i-1} A w_{j+1}^n$.

(These conditions together imply that $S \Rightarrow^* w$.)

We argue, as do Ruzzo [1979] and, for a different formalism, Satta [1994], that a practical parser must create output from which c-derivation information can be retrieved efficiently. This information is what allows us to ascertain that there exists an analysis of the input sequence for which a certain substring forms a *constituent*, or coherent unit. In contrast, derivation information records potential subderivations that may not be consistent with any analysis of the full input string. For example, in the sentence "Only the lonely can play," "the lonely can" could conceivably, in isolation, form a noun phrase, but clearly, in any reasonable grammar of English, no nonterminal c-derives that substring. While some parsers retain information about derivations that are not c-derivations, we formulate our definition of parsing to include algorithms that do not.

Definition 2. A *c-parser* is an algorithm that takes a *CFG* $G = (\Sigma, V, R, S)$ and string $w \in \Sigma^*$ as input and produces output $\mathcal{F}_{G,w}$ that acts as an oracle about parse information as follows: for any $A \in V$,

- If A *c*-derives w_i^j, then $\mathcal{F}_{G,w}(A, i, j) =$ "yes."
- If $A \not\Rightarrow^* w_i^j$ (which implies that A does not *c*-derive w_i^j), then $\mathcal{F}_{G,w}(A, i, j) =$ "no."
- $\mathcal{F}_{G,w}$ answers queries in constant time.

The asymmetry of derivation and c-derivation in our definition of c-parsing is deliberate. We allow $\mathcal{F}_{G,w}$'s answer to be arbitrary if $A \Rightarrow^* w_i^j$ but A does not c-derive w_i^j; we leave it to the algorithm designer to decide which answer is appropriate. Thus, our definition makes the class of c-parsers as broad as possible: if we had changed the first condition to "If A derives $w_i^j \cdots$", then Earley parsers would be excluded, since they do not keep track of all substring derivations; whereas if we had written the second condition as "If A does not c-derive $w_i^j, \ldots$", then CKY would not be a c-parser, since it tracks all substring derivations, not just c-derivations. In fact, the class of c-parsers contains all *tabular* parsers, including generalized LR parsing, CKY, and Earley's algorithm [Nederhof and Satta 1996]. In contrast, Ruzzo [1979] deals with the difference between derivations and c-derivations by defining two different problems (the *weak all-parses problem* and the *all-parses problem*).

Our choice of an oracle rather than a specific data structure as the output of a c-parser is also for the purpose of keeping our definition as broad as possible. In tabular algorithms like CKY, the oracle is given in the form of a matrix or *chart*; indeed, Ruzzo's [1979] definition of the all-parses and weak all-parses problems requires the output to be a matrix. However (as Ruzzo points out), this is not the only possibility, and furthermore has a liability from a technical point of view: if the output must be a matrix, then all parsing algorithms must take time at least $\Omega(n^2)$ even to print their output. Since it may be possible for c-derivations to be represented more compactly, we prefer to allow for this possibility in our definition.

Finally, with regards to the third condition, we observe that Satta [1994] imposes the same constant-time constraint for a different grammar formalism (tree-adjoining grammars). On the other hand, we could loosen this to allow query processing to take time polylogarithmic in the string and grammar size without much effect on our results (see Section 3.5).

2.3. ANALYZING PARSER RUNTIMES. It is common in the formal language theory literature to see the running time of parsing algorithms described as a function of the length of the input string only (e.g., $O(n^3)$ for a string of length n). That is, the size of the context-free grammar is often treated as a constant. This stems in part from two characteristics of the programming languages and compilers domains: first, the size of a computer program's source code is typically much greater than the size of the grammar describing the programming language's syntax, so that the grammar term is negligible; and second, compilers are constructed to analyze many different programs with respect to a single built-in grammar.

However, in other domains, these conditions do not hold. For example, in natural language, sentences are relatively short (not often longer than one hundred words) compared with the size of the grammar: Johnson [1998] describes a (probabilistic) CFG for a subset of English that has 22,773 rules. Indeed, Joshi[1997] notes that "the real limiting factor in practice is the size of the grammar." Therefore, it is reasonable to include in the analysis of parsing time the dependence on the grammar size, and we will do so here. As a point of information, we note that both CKY and Earley's algorithm can be implemented to run in time $O(|G|n^3)$ [Graham et al. 1980], although CKY requires the input grammar to be in Chomsky normal form, conversion to which may cause a quadratic increase in the number of productions in the grammar [Hopcroft and Ullman 1979].

3. *The Reduction*

In this section, we provide two efficient reductions of Boolean matrix multiplication to c-parsing, thus proving that any c-parsing algorithm can be used as a Boolean matrix multiplication algorithm with little computational overhead. The first reduction produces a string and a context-free grammar; the second is a modification of the first in which the grammar produced is in Chomsky normal form. The techniques we use are an adaptation of Satta's [1994] elegant reduction of Boolean matrix multiplication to *tree-adjoining grammar* (TAG) parsing. However, Satta's results rely explicitly on properties of TAGs that allow them to generate non-context-free languages, and so cannot be directly applied to CFGs.

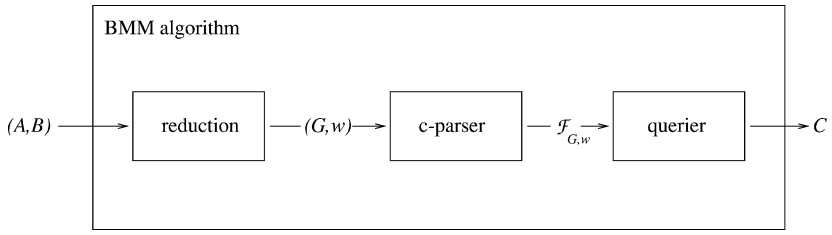

FIG. 3. Converting a c-parser into a BMM algorithm.

3.1. BOOLEAN MATRIX MULTIPLICATION. A Boolean matrix is a matrix with entries from the set $\{0, 1\}$. A Boolean matrix multiplication (BMM) algorithm takes as input two $m \times m$ Boolean matrices A and B and returns their *Boolean product* $A \times B$, which is the $m \times m$ Boolean matrix C whose entries are defined by

$$c_{ij} = \bigvee_{k=1}^{m} (a_{ik} \wedge b_{kj}).$$

That is, $c_{ij} = 1$ if and only if there exists a number k, $1 \leq k \leq m$, such that $a_{ik} = b_{kj} = 1$.

As noted above, the Boolean product C can be computed via standard matrix multiplication, since $c_{ij} = \sum_{k=1}^{m} a_{ik} \cdot b_{kj}$. This means that we can use the Coppersmith and Winograd [1990] general matrix multiplication algorithm to calculate the Boolean matrix product of two $m \times m$ Boolean matrices in time $O(m^{2.376})$. To our knowledge, the asympotically fastest algorithms for BMM all rely on general matrix multiplication; the fastest algorithms that do not do so are the so-called "four Russians" algorithm [Arlazarov et al. 1970], with worst-case running time $O(m^3/\log(m))$, and Rytter's [1985] variant, which uses compression to reduce the time to $O(m^3/\log^2(m))$.

3.2. THE REDUCTION: FIRST VERSION. Our goal in this section is to show that Boolean matrix multiplication can be efficiently reduced to c-parsing of CFGs. That is, we describe a simple procedure that takes as input an instance of the BMM problem and converts it into an instance of the CFG parsing problem with the following property: any c-parsing algorithm run on the new parsing problem yields output from which it is easy to determine the answer to the original BMM problem. We therefore demonstrate that any c-parser can be used to solve Boolean matrix multiplication via the three-step process shown schematically in Figure 3.

Thus, given two Boolean matrices A and B, we need show how to produce a grammar G and a string w such that c-parsing w with respect to G yields output $\mathcal{F}_{G,w}$ from which information about the Boolean product $C = A \times B$ can be easily retrieved. Our approach will be to encode almost all the information about A and B in the grammar.

We can sketch the desired behavior of the grammar G as follows: Suppose entries a_{ik} in A and b_{kj} in B are both 1. Assume we have some way to break up array indices into two parts so that i can be reconstructed from i_1 and i_2, j can be reconstructed from j_1 and j_2, and k can be reconstructed from k_1 and k_2 (we will describe a way to do this later; the motivation is to keep the grammar size relatively small). Then,

our grammar will permit the following derivation sequence:

$$\begin{aligned} C_{i_1,j_1} &\Rightarrow A_{i_1,k_1} B_{k_1,j_1} \\ &\Rightarrow^* \underbrace{w_{i_2} \cdots w_{k_2+\delta}}_{\text{derived by } A_{i_1,k_1}} \underbrace{w_{k_2+\delta+1} \cdots w_{j_2+2\delta}}_{\text{derived by } B_{k_1,j_1}}, \end{aligned}$$

where δ will be defined later. The key thing to observe is that C_{i_1,j_1} generates two nonterminals whose "inner" indices match, and that these two nonterminals generate substrings that lie exactly next to each other. The "inner" indices constitute a check on k_1, and substring adjacency constitutes a check on k_2; together, these two checks serve as a proof that $a_{ik} = b_{kj} = 1$, and hence that c_{ij} is also 1.

We now set up some notation. Let A and B be two Boolean matrices, each of size $m \times m$, and let C be their Boolean matrix product. In the rest of this section, we consider A, B, C, and m to be fixed. Set $d = \lceil m^{1/3} \rceil$, and set $\delta = d + 2$. (The effect of these choices on the efficiency of our reduction is discussed in Section 3.5.) We will be constructing a string of length 3δ; we choose δ slightly larger than d in order to avoid having epsilon-productions in our grammar.

Our index encoding function is as follows: Let i be a matrix index, $1 \le i \le m \le d^3$. Then, we define the function $f(i) = (f_1(i), f_2(i))$ by

$$\begin{aligned} f_1(i) &= \lfloor i/d \rfloor \qquad\qquad (\text{so that } 0 \le f_1(i) \le d^2), \quad \text{and} \\ f_2(i) &= (i \bmod d) + 2 \quad (\text{so that } 2 \le f_2(i) \le d+1). \end{aligned}$$

Since $f_1(i)$ and $f_2(i)$ are essentially the quotient and remainder of integer division of i by d, we can reconstruct i from $(f_1(i), f_2(i))$. It may be helpful to think of these two quantities as "high-order" and "low-order" bits, respectively. For convenience, we will employ the notational shorthand of using subscripts instead of the functions f_1 and f_2; that is, we write i_1 and i_2 for $f_1(i)$ and $f_2(i)$.

It is now our job to create a CFG $G = (\Sigma, V, R, S)$ and a string $w \in \Sigma^*$ that encode information about A and B and express constraints about their product C.

We choose the set of terminals to be $\Sigma = \{w_\ell : 1 \le \ell \le 3d + 6\}$. The string we choose is extremely simple, and in fact doesn't depend on A or B at all: we set $w = w_1 w_2 \cdots w_{3d+6}$. We consider w to be made up of three parts, x, y, and z, each of size δ:

$$w = \underbrace{w_1 w_2 \cdots w_{d+2}}_{x} \underbrace{w_{d+3} \cdots w_{2d+4}}_{y} \underbrace{w_{2d+5} \cdots w_{3d+6}}_{z}.$$

Observe that for any array index i between 1 and m, it is the case that w_{i_2} appears in x, $w_{i_2+\delta}$ appears in y, and $w_{i_2+2\delta}$ appears in z, since

$$\begin{aligned} i_2 &\in [2, d+1], \\ i_2 + \delta &\in [d+4, 2d+3], \text{ and} \\ i_2 + 2\delta &\in [2d+6, 3d+5]. \end{aligned}$$

We now turn our attention to constructing the grammar G. Our plan is to include a set of nonterminals $\{C_{p,q} : 1 \le p, q \le d^2\}$ in V such that $c_{ij} = 1$ if and only if C_{i_1,j_1} c-derives $w_{i_2}^{j_2+2\delta}$.

3.3. THE GRAMMAR. To create $G = (\Sigma, V, R, S)$, we build up the set of nonterminals and productions, starting with $V = \{S\}$ and $R = \emptyset$. We add nonterminal

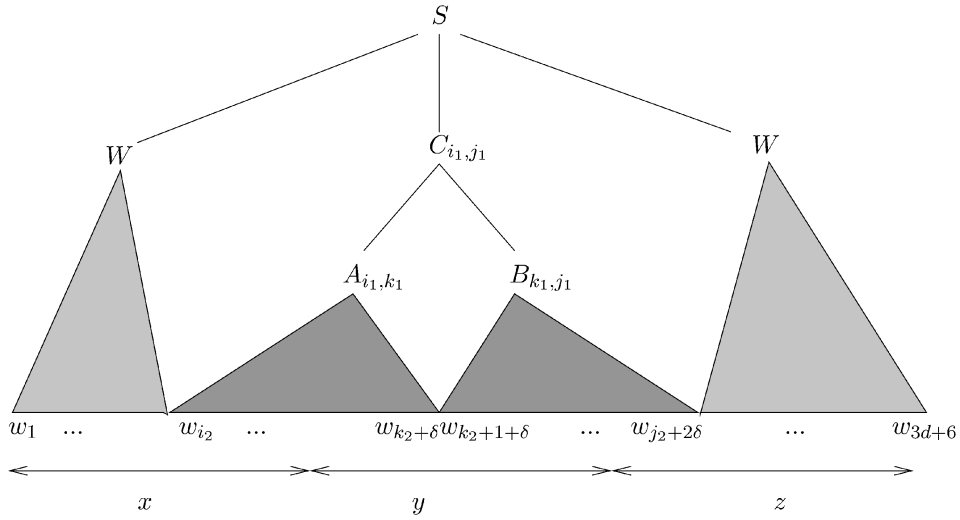

FIG. 4. Schematic of the derivation process when $a_{ik} = b_{kj} = 1$. The substrings derived by A_{i_1,k_1} and B_{k_1,j_1} lie right next to each other.

W to V for generating arbitrary nonempty substrings and therefore add productions

$$W \to w_\ell W | w_\ell, \quad 1 \le \ell \le 3d+6. \qquad (W\text{-rules})$$

Next, we encode the entries of the input matrices A and B in our grammar. We add the nonterminals from the sets $\{A_{p,q} : 1 \le p, q \le d^2\}$ and $\{B_{p,q} : 1 \le p, q \le d^2\}$. Then, for every *nonzero* entry a_{ij} in A, we add the production

$$A_{i_1,j_i} \to w_{i_2} W w_{j_2+\delta}. \qquad (A\text{-rules})$$

For every *nonzero* entry b_{ij} in B, we add the production

$$B_{i_1,j_i} \to w_{i_2+1+\delta} W w_{j_2+2\delta}. \qquad (B\text{-rules})$$

To represent the entries of C, we add the nonterminals from the set $\{C_{p,q} : 1 \le p, q \le d^2\}$ and include productions

$$C_{p,q} \to A_{p,r} B_{r,q}, \quad 1 \le p, q, r \le d^2. \qquad (C\text{-rules})$$

Finally, we complete the construction with productions for the start symbol S:

$$S \to W C_{p,q} W, \quad 1 \le p, q \le d^2. \qquad (S\text{-rules})$$

We now prove the following result about the grammar and string we have just described.

THEOREM 1. *For $1 \le i, j \le m$, the entry c_{ij} in C is nonzero if and only if C_{i_1,j_1} c-derives $w_{i_2}^{j_2+2\delta}$.*

PROOF. Fix i and j.

Let us prove the "only if" direction first. Thus, suppose $c_{ij} = 1$. Then there exists a k such that $a_{ik} = b_{kj} = 1$. Figure 4 sketches how C_{i_1,j_1} c-derives $w_{i_2}^{j_2+2\delta}$.

CLAIM 1. $C_{i_1,j_1} \Rightarrow^* w_{i_2}^{j_2+2\delta}$.

The production $C_{i_1,j_1} \rightarrow A_{i_1,k_1} B_{k_1,j_1}$ is one of the C-rules in our grammar. Since $a_{ik} = 1$, $A_{i_1,k_1} \rightarrow w_{i_2} W w_{k_2+\delta}$ is one of our A-rules, and since $b_{kj} = 1$, $B_{k_1,j_1} \rightarrow w_{k_2+1+\delta} W w_{j_2+2\delta}$ is one of our B-rules. Finally, since $i_2 + 1 < (k_2 + \delta) - 1$ and $(k_2 + 1 + \delta) + 1 \leq (j_2 + 2\delta) - 1$, we have $W \Rightarrow^* w_{i_2+1}^{k_2+\delta-1}$ and $W \Rightarrow^* w_{k_2+2+\delta}^{j_2+2\delta-1}$, since both substrings are of length at least one. Therefore,

$$\begin{aligned} C_{i_1,j_1} &\Rightarrow A_{i_1,k_1} B_{k_1,j_1} \\ &\Rightarrow^* \underbrace{w_{i_2} W w_{k_2+\delta}}_{\text{derived by } A_{i_1,k_1}} \underbrace{w_{k_2+1+\delta} W w_{j_2+2\delta}}_{\text{derived by } B_{k_1,j_1}} \\ &\Rightarrow^* w_{i_2}^{j_2+2\delta}. \end{aligned}$$

□

CLAIM 2. $S \Rightarrow^* w_1^{i_2-1} C_{i_1,j_1} w_{j_2+2\delta+1}^{3d+6}$.

This claim is essentially trivial, since by the definition of the S-rules, we know that $S \Rightarrow^* W C_{i_1,j_1} W$. We need only show that neither $w_1^{i_2-1}$ nor $w_{j_2+2\delta+1}^{3d+6}$ is the empty string (and hence can be derived by W); since $1 \leq i_2 - 1$ and $j_2 + 2\delta + 1 \leq 3d + 6$, the claim holds. □

Claims 1 and 2 together prove that C_{i_1,j_1} c-derives $w_{i_2}^{j_2+2\delta}$, as required.[2]

Next we prove the "if" direction. Suppose C_{i_1,j_1} c-derives $w_{i_2}^{j_2+2\delta}$, which by definition means $C_{i_1,j_1} \Rightarrow^* w_{i_2}^{j_2+2\delta}$. This can only arise through the application of a C-rule:

$$C_{i_1,j_1} \Rightarrow A_{i_1,k'} B_{k',j_1} \Rightarrow^* w_{i_2}^{j_2+2\delta}$$

for some k'. It must be the case that, for some ℓ, $A_{i_1,k'} \Rightarrow^* w_{i_2}^{\ell}$ and $B_{k',j_1} \Rightarrow^* w_{\ell+1}^{j_2+2\delta}$. But then we must have the productions $A_{i_1,k'} \rightarrow w_{i_2} W w_{\ell}$ and $B_{k',j_1} \rightarrow w_{\ell+1} W w_{j_2+2\delta}$ with $\ell = k'' + \delta$ for some k''. But we can only have such productions if there exists a number k such that $k_1 = k'$, $k_2 = k''$, $a_{ik} = 1$, and $b_{kj} = 1$; and this implies that $c_{ij} = 1$. □

Examination of the proof reveals that we also have the following two corollaries.

COROLLARY 1. *For $1 \leq i, j \leq m$, $c_{ij} = 1$ if and only if $C_{i_1,j_1} \Rightarrow^* w_{i_2}^{j_2+2\delta}$. Hence, c-derivation and derivation are equivalent for the $C_{p,q}$ nonterminals.*

COROLLARY 2. *$S \Rightarrow^* w$ if and only if C is not the all-zeroes matrix.*

Let us now calculate the size of G. V consists of roughly $3((d^2)^2) \approx m^{4/3}$ nonterminals. R contains about $6d$ W-rules and $(d^2)^2 \approx m^{4/3}$ S-rules. There are at most m^2 A-rules, since we have A-rules only for each nonzero entry in A; similarly, there are at most m^2 B-rules. And lastly, there are $(d^2)^3 \approx m^2$ C-rules. Therefore, our grammar is of size $O(m^2)$ with a very small constant factor; considering

[2] This proof would have been simpler if we had allowed W to derive the empty string. However, we avoid epsilon-productions in order to facilitate the conversion to Chomsky normal form discussed in the next section.

$$
\begin{array}{lcll}
W & \longrightarrow & W_\ell W | w_\ell & (1 \le \ell \le 3d+6) \\
W_\ell & \longrightarrow & w_\ell & (1 \le \ell \le 3d+6) \\
 & & & \\
A_{i_1,j_1} & \longrightarrow & W_{i_2} X_{j_2+\delta} & \text{(one for each nonzero entry } a_{ij} \text{ in } A) \\
X_{j_2+\delta} & \longrightarrow & W W_{j_2+\delta} & (2 \le j_2 \le d+1) \\
 & & & \\
B_{i_1,j_1} & \longrightarrow & W_{i_2+1+\delta} X_{j_2+2\delta} & \text{(one for each nonzero entry } b_{ij} \text{ in } B) \\
X_{j_2+2\delta} & \longrightarrow & W W_{j_2+2\delta} & (2 \le j_2 \le d+1) \\
 & & & \\
C_{p,q} & \longrightarrow & A_{p,r} B_{r,q} & (1 \le p,q,r \le d^2) \\
 & & & \\
S & \longrightarrow & WT & \\
T & \longrightarrow & C_{p,q} W & (1 \le p,q \le d^2)
\end{array}
$$

FIG. 5. A Chomsky normal form version of the productions of the grammar from the previous section.

that G encodes $m \times m$ matrices A and B, it is not possible to shrink this much further.

3.4. CHOMSKY NORMAL FORM. We would like our results to cover as large a class of parsers as possible. Some parsers, such as CKY, require the input grammar to be in Chomsky normal form (CNF), that is, where the right-hand side of every production consists of either exactly two nonterminals or exactly a single terminal. We therefore wish to construct a CNF version G' of G. However, not only do we want Theorem 1 to hold for G' as well as G, but, in order to preserve time bounds, we also desire that $|G'| = O(|G|)$.

Unfortunately, the standard algorithm for converting CFGs to CNF can yield a quadratic blow-up in the number of productions in the grammar [Hopcroft and Ullman 1979] and thus is clearly unsatisfactory for our purposes. However, since G contains no epsilon-productions or unit productions, it is easy to convert G by adding a small number of record-keeping nonterminals and productions, with the resultant grammar G' having very similar parse trees—in particular, the set of substrings that are c-derived by the $C_{p,q}$ nonterminals are the same in each grammar. Figure 5 gives the productions of G'. Note that G' has only $O(d)$ more productions and nonterminals, and so $|G'| = O(m^2)$ as well.

3.5. TIME BOUNDS. We are now in a position to show the relation between time bounds for Boolean matrix multiplication and time bounds for CFG parsing.

THEOREM 2. *Any c-parser P with running time $O(T(g)t(n))$ on grammars of size g and strings of length n can be converted into a BMM algorithm M_P that runs in time $O(max(m^2, T(m^2)t(m^{1/3})))$. In particular, if P takes time $O(gn^{3-\epsilon})$, then M_P runs in time $O(m^{3-\epsilon/3})$.*

PROOF. M_P acts as sketched in Figure 3. More precisely, given two Boolean $m \times m$ matrices A and B, it constructs G (or G', as required) and w as described above. It feeds G and w to P, which outputs oracle $\mathcal{F}_{G,w}$. To compute the product matrix C, M_P requests from the oracle the value of $\mathcal{F}_{G,w}(C_{i_1,j_1}, i_2, j_2 + 2\delta)$

(i.,e., whether or not C_{i_1,j_1} derives or c-derives[3] $w_{i_2}^{j_2+2\delta}$) for each i and j, $1 \leq i, j \leq m$, setting c_{ij} to one if and only if the answer is "yes."

The running time of M_P is computed as follows: It takes $O(m^2)$ time to read the two input matrices. Since G is of size $O(m^2)$ and $|w| = O(m^{1/3})$, it takes $O(m^2)$ time to build the input to P, which then computes $\mathcal{F}_{G,w}$ in time $O(T(m^2)t(m^{1/3}))$. Retrieving C takes $O(m^2)$ since, by definition of c-parser, each query to the oracle takes constant time. So the total time spent by M_P is $O(\max(m^2, T(m^2)t(m^{1/3})))$, as claimed.

Note that if we redefine c-parsing so that oracle queries take $f(g, n)$ time instead of constant time, where g is the size of the grammar and n is the length of the string, then the bound changes to $O(\max(m^2 f(g, n), T(m^2)t(m^{1/3})))$; as long as f is polylogarithmic, the second argument of the maximum in the bound surely dominates.

In the case where $T(g) = g$ and $t(n) = n^{3-\epsilon}$, M_P has a running time of $O(m^2(m^{1/3})^{3-\epsilon}) = O(m^{3-\epsilon/3})$. □

The case in which P takes time linear in the grammar size is of the most interest, since, as mentioned above, in natural language processing applications the grammar tends to be far larger than the strings to be parsed. In this case, our result directly converts any improvement in the exponent for CFG parsing to a reduction in the exponent for BMM.

3.5.1. *Parameter Choices.* Since Valiant [1975] proved that an $O(m^{3-\epsilon})$ BMM algorithm can be transformed into a parser with time complexity $O(n^{3-\epsilon})$ in the string length, it is natural to ask whether our technique could yield the stronger result (if it is in fact true) that a CFG parser running in time $O(gn^{3-\epsilon})$ can be converted into an $O(m^{3-\epsilon})$ BMM algorithm. We now explain why such a result cannot be obtained by a straightforward modification of the reduction method we described above.

Our run-time results are based on a particular choice of where to divide matrix indices into "high-order bits" and "low-order bits"; in particular, we set d, which parametrizes the number of low-order bits, to $d = \lceil m^{1/3} \rceil$. We determined this value by considering the effect of d on the size of the resulting grammar and string: roughly speaking, a larger value shrinks the former but expands the latter. For convenience, let us set $d = m^\ell$, and consider how to pick ℓ.

Since combining the higher-order bits and the lower-order bits yields a matrix index of magnitude at most m, it follows that the string has size $O(m^\ell)$ and the grammar will have size $O(m^2 + (m^{1-\ell})^3)$ (the first term comes from the inclusion of the A- and B-rules, and the second term comes from the fact that the C-rules have to include the higher-order bits for three matrix indices). Hence, a parser with run-time complexity $O(gn^{3-\epsilon})$ yields a BMM algorithm with run-time complexity $O(m^{2+(3-\epsilon)\ell} + m^{3-\epsilon\ell})$. Inspection reveals that, when $\ell > 1/3$, the first term dominates; when $\ell < 1/3$, the second term dominates; and the lowest upper bound occurs at the "crossing point" where $\ell = 1/3$.

4. *Related Results*

We have shown that the existence of a fast practical CFG parsing algorithm would yield a fast practical BMM algorithm. Given that fast practical BMM

[3]By Corollary 1, the two notions are equivalent in this case.

algorithms are thought not to exist, this establishes a limitation on the efficiency of practical CFG parsing, and helps explain why there has been very little success in developing practical subcubic general CFG parsers.

There have been a number of related results regarding the time complexity of context-free grammar parsing and the relationship between this and other problems. We survey these results below.

As mentioned above, the asymptotically fastest (although not practical) general context-free parsing algorithm is due to Valiant [1975], who showed that the problem can be reduced to Boolean matrix multiplication (this is the "opposite direction" of the reduction we present). His algorithm shows that the worst-case dependence of the speed of CFG parsing on the input string length is $O(M(n))$, where $M(m)$ is the time it takes to multiply two $m \times m$ Boolean matrices together. (Rytter [1995] provides an alternate version of this algorithm with the same asymptotic complexity.)

Methods for reducing Boolean matrix multiplication to context-free grammar parsing were previously considered by Ruzzo [1979]. He proved that the problem of producing all possible parses of a string of length n with respect to a context-free grammar is at least as hard as multiplying two $\sqrt{n} \times \sqrt{n}$ Boolean matrices together. His technique encodes most of the information about the matrices in strings (as opposed to in the grammar, as in our method). Ruzzo's result does not serve to explain why practical subcubic CFG parsing algorithms have been so difficult to produce, since using his reduction translates even a parser running in time proportional to $n^{1.5}$ to a cubic-time BMM algorithm.

Harrison and Havel [Harrison and Havel 1974; Harrison 1978] note that there is a reduction of $m \times m$ BMM *checking* to context-free *recognition* (a BMM checker takes as input three Boolean matrices A, B, and C and reveals whether or not C is the Boolean product of A and B). These two decision problems are clearly related to the algorithmic problems we consider in this article. However, this reduction, like Ruzzo's, also converts a parser running in time proportional to $n^{1.5}$ to a cubic-time BMM checking algorithm, which, again, is not as strong a result as ours.

The problem of *on-line* CFL recognition is to proceed through each prefix w_1^i of the input string w, determining whether or not w_1^i is generated by the input context-free grammar before reading the next $((i+1)\text{th})$ input symbol. The study of the complexity of this problem has a long history; in fact, the landmark paper of Hartmanis and Stearns [1965] that introduced the notions of time and space complexity contains an example of a CFL for which on-line recognition of strings of length n takes more than n steps. Currently, the best known lower bound for this problem is $\Omega(n^2/\log n)$ [Seiferas 1986; Gallaire 1969]. However, on-line recognition is a more difficult task than the standard CFL recognition problem (indeed, it is the extra constraints imposed by the on-line requirement that make it easier to prove lower bounds), and so these results do not translate to the usual recognition paradigm. To date, there are no nontrivial lower bounds known for general CFL recognition.

Relationships between parsing other grammatical formalisms and multiplying Boolean matrices have also been explored. In particular, several researchers have looked at *Tree Adjoining Grammar* (TAG) [Joshi et al. 1975], an elegant formalism based on modifying tree structures. TAGs have strictly greater generative capacity than context-free grammars, but at the price of being (apparently) harder

to parse: standard algorithms run in time proportional to n^6, although Rajasekaran and Yooseph [1998] adapt Valiant's [1975] technique to get an asymptotically faster parser using BMM. Satta [1994] gives a reduction of Boolean matrix multiplication to tree-adjoining grammar parsing, demonstrating that any substantial improvement over $O(gn^6)$ for TAG parsing would result in a subcubic BMM algorithm. Our reduction was inspired by Satta's and resembles his in the way that matrix information is encoded in a grammar. However, Satta's reduction explicitly relies on TAG properties that allow non-context-free languages to be generated, and so cannot be directly applied to CFG parsing.

ACKNOWLEDGMENTS. Thanks to Zvi Galil, Joshua Goodman, Rebecca Hwa, Jon Kleinberg, Giorgio Satta, Stuart Shieber, Les Valiant, and the anonymous referees for many helpful comments and conversations.

REFERENCES

AHO, A. V., SETHI, R., AND ULLMAN, J. D. 1986. *Compilers: Principles, Techniques and Tools*. Addison-Wesley, Reading, Mass.

ARLAZAROV, V. L., DINIC, E. A., KRONROD, M. A., AND FARADŽEV, I. A. 1970. On economical construction of the transitive closure of an oriented graph. *Soviet Math. Dokl. 11*, 1209–1210. (English translation of Russian article in *Dokl. Akad. Nauk SSSR 194* (1970).)

BAILEY, D. 1988. Extra high speed matrix multiplication on the Cray-2. *SIAM J. Sci. Stat. Comput. 9*, 3, 603–607.

CHOMSKY, N. 1956. Three models for the description of language. *IRE Trans. Inf. Theory 2*, 3, 113–124.

COPPERSMITH, D., AND WINOGRAD, S. 1987. Matrix multiplication via arithmetic progression. In *Proceedings of the ACM Annual Symposium on the Theory of Computing*. ACM, New York, pp. 1–6.

COPPERSMITH, D., AND WINOGRAD, S. 1990. Matrix multiplication via arithmetic progression. *J. Symb. Comput. 9*, 3, 251–280 (Special Issue on Computational Algebraic Complexity).

DURBIN, R., EDDY, S., KROGH, A., AND MITCHISON, G. 1998. *Biological Sequence Analysis*. Cambridge University Press, Cambridge, Mass.

EARLEY, J. 1970. An efficient context-free parsing algorithm. *Communi. ACM 13*, 2 (Feb.), 94–102.

GALLAIRE, H. 1969. Recognition time of context-free languages by on-line Turing machines. *Inf. Cont. 15*, 3 (Sept.), 288–295.

GRAHAM, S. L., HARRISON, M. A., AND RUZZO, W. L. 1980. An improved context-free recognizer. *ACM Trans. Prog. Lang. Syst. 2*, 3, 415–462.

HARRISON, M., AND HAVEL, I. 1974. On the parsing of deterministic languages. *J. ACM 21*, 4 (Oct.), 525–548.

HARRISON, M. A. 1978. *Introduction to Formal Language Theory*. Addison-Wesley, Reading, Mass.

HARTMANIS, J., AND STEARNS, R. E. 1965. On the computational complexity of algorithms. *Trans. AMS 117*, 285–306.

HOPCROFT, J. E., AND ULLMAN, J. D. 1979. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, Mass.

JOHNSON, M. 1998. PCFG models of linguistic tree representations. *Computat. Ling. 24*, 4, 613–632.

JOSHI, A. 1997. Parsing techniques. In *Survey of the State of the Art in Human Language Technology*, Ronald Cole, Joseph Mariani, Hans Uszkoreit, Giovanni Battista Varile, Annie. Zaenen, Antonio. Zampolli, and Victor Zue, Eds. Studies in Natural Language Processing. Cambridge University Press, Cambridge, Mass, pp. 351–356.

JOSHI, A. K., LEVY, L. S., AND TAKAHASHI, M. 1975. Tree adjunct grammars. *J. Comput. Syst. Sci. 10*, 1, 136–163.

JURAFSKY, D., AND MARTIN, J. H. 2000. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall series in Artificial Intelligence. Prentice-Hall, Englewood Cliffs, N. J.

KASAMI, T. 1965. An efficient recognition and syntax algorithm for context-free languages. Scientific Rep. AFCRL-65-758. Air Force Cambridge Research Lab, Bedford, Mass.

NEDERHOF, M. J., AND SATTA, G. 1996. Efficient tabular LR parsing. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics*. pp. 239–246.

RAJASEKARAN, S., AND YOOSEPH, S. 1998. TAL recognition in $O(M(n^2))$ time. *J. Comput. Syst. Sci. 56*, 1, 83–89.

RUZZO, W. L. 1979. On the complexity of general context-free language parsing and recognition. In *Proceedings of the 6th Colloquium on Automata, Languages and Programming* (*ICALP*), Lecture Notes in Computer Science, vol. 71. Springer-Verlag, New York, pp. 489–497.

RYTTER, W. 1985. Fast recognition of pushdown automaton and context-free languages. *Inf. Cont. 67*, 12–22.

RYTTER, W. 1995. Context-free recognition via shortest paths computation: A version of Valiant's algorithm. *Theoret. Comput. Sci. 143*, 2, 343–352.

SATTA, G. 1994. Tree-adjoining grammar parsing and Boolean matrix multiplication. *Comput. Ling. 20*, 2, 173–191.

SEIFERAS, J. 1986. A simplified lower bound for context-free-language recognition. *Inf. Cont. 69*: 255–260.

STRASSEN, V. 1969. Gaussian elimination is not optimal. *Num. Math. 14*, 3, 354–356.

STRASSEN, V. 1990. Algebraic complexity theory. In *Handbook of Theoretical Computer Science*, vol. A, Jan van Leeuwen, Ed. Elsevier Science Publishers, Amsterdam, The Netherlands, pp. 633–672.

THOTTETHODI, MITHUNA, T., CHATTERJEE, S., AND LEBECK, A. R. 1998. Tuning Strassen's matrix multiplication for memory efficiency. In *SC98: High Performance Networking and Computing Conference*.

VALIANT, L. G. 1975. General context-free recognition in less than cubic time. *J. Comput. Syst. Sci. 10*, 308–315.

YOUNGER, D. H. 1967. Recognition and parsing of context-free languages in time n^3. *Inf. Cont. 10*, 2, 189–208.

RECEIVED FEBRUARY 2001; REVISED DECEMBER 2001; ACCEPTED DECEMBER 2001