CFG Parsing and Boolean Matrix Multiplication

Franziska Ebert

Abstract. In this work the relation between Boolean Matrix Multiplication (BMM) and Context Free Grammar (CFG) parsing is shown. The first described approach, which is due to Valiant (1975), shows how CFG parsing can be reduced to Boolean Matrix Multiplication. Afterwards the reverse direction, i.e. how a CFG parser can be used to multiply two Boolean matrices, is presented, which is due to Lee (2002). The fundamental theorem that can be derived from the reductions is that fast CFG parsing requires fast Boolean matrix multiplication, and vice versa.

1 Introduction

CFG parsing arose in the middle of the last century. First CFG parsers had a worst-case running time of $O(gn^3)$, where g is the size of the CFG and n is the length of the input string. The most famous CFG parsers with this running time are the CKY algorithm ([You67], 1967) and Earley's algorithm ([Ear70], 1970).

A relation between CFG parsing and Boolean Matrix Multiplication (BMM) was found at first by Valiant in 1975 ([Val75]). Since BMM was shown to be subcubic (Strassen: $O(n^{2.81})$, [Str69]), Valiant tried to transform the CFG parsing problem to an instance for BMM with no computational overhead. Indeed, his algorithm is proven to have a worst-case running time in $O(BM(n))$, where $BM(m)$ is the time needed to multiply two $m \times m$ Boolean matrices. Hence, if Valiant's algorithm uses Strassen's method, CFG parsing is possible in less than cubic time. Also today Valiant's algorithm is asymptotically the best.

During the following years, there were only little improvements of the running time of BMM algorithms. Unfortunately, it turns out that the involved constants in these "fast" BMM algorithms are very large, such that these algorithms cannot be used in practice. Therefore, still Strassen's algorithm has asymptotically the best running time.

For the development of CFG parsers it looks almost the same. There was only little success in finding a better algorithm. Thus, Lee analyzes the relation between BMM and CFG parsing ([Lee02]). She shows that the reverse direction of Valiant's approach, namely converting a CFG parser into an algorithm for BMM, is also possible. Figure 1 gives an overview over the development of BMM algorithms (below the timeline) and CFG parsing algorithms (above the timeline).

Since Valiant's reduction and also the reverse reduction could be proven, Lee formulated a fundamental theorem in 2002: "Fast CFG parsing requires fast Boolean matrix multiplication." ([Lee02]). In the following this theorem of Lee is proven by showing the reduction of CFG parsing to BMM and the reverse direction.

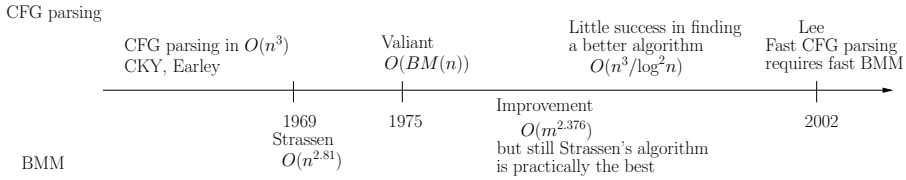

Fig. 1. Development of CFG parsing algorithms and BMM algorithms.

2 Reduction of CFG Parsing to Boolean Matrix Multiplication

The idea of reducing CFG parsing to BMM is due to Valiant ([Val75]). In his paper of 1975 he derives a sub-cubic CFG parsing algorithm which - also today - is the asymptotically fastest known. First, CFG parsing is reduced to matrix multiplication (MM) and afterwards MM is reduced to BMM. In the following this CFG parsing algorithm is described.

The general idea for an MM algorithm to parse a string of length n and a CFG is to build an upper triangular $(n+1)\times(n+1)$ matrix a with subsets of the set of nonterminals as elements. Matrix multiplication is defined newly over this type of matrices. After applying this new matrix multiplication several times on a, a fixpoint is reached. In all iteration steps it holds that the entry a_{ij} of a is the set of nonterminals which derive the string w_i^{j-1}. Hence, if the starting symbol is an element of the set a_{1n+1} of matrix a, the string can be derived by the CFG. Note, that the form of a is exactly like the recognition matrix of the CKY-algorithm. Now Valiant's CFG parsing algorithm is formalized.

2.1 Preliminaries

It is assumed that the considered CFGs are in Chomsky normal form. Thus CFGs are defined as follows.

Definition 1. *A CFG is a 4-tuple* (N, Σ, P, A_1) *with*

- N*: The set of nonterminals, with* $N = \{A_1, \ldots, A_h\}$
- Σ*: The set of terminals*
- P*: The set of productions, each of which has the form:*
  - $A_i \rightarrow A_j A_k$
  - $A_i \rightarrow x$*, for* $x \in \Sigma$
- A_1*: The starting symbol*

With $A_i \rightarrow^* w_j^k$ is denoted that the string $w_j \ldots w_k$ can be derived from A_i.

In Valiant's algorithm, matrices with subsets of N as elements are observed. Hence, the MM has to be defined newly.

Definition 2. *Let a and b be two $m \times m$ matrices with subsets of N as elements. Then $a * b = c$ is defined as:*

$$c_{ij} = \bigcup_{k=1}^{m} a_{ik} * b_{kj} \qquad \forall 1 \leq i, j \leq m$$

where the $*$-operator over subsets of N is defined as:

$$N_1 * N_2 = \{A_i \mid \exists A_j \in N_1, A_k \in N_2.(A_i \rightarrow A_j A_k) \in P\}$$

Confer example 1 for an application of multiplying two matrices with subsets of N as elements.

Example 1. Consider the CFG G with starting symbol S and productions
$P =$
$\{\ S \rightarrow XY$
$X \rightarrow XA \mid AA$
$Y \rightarrow YB \mid BB$
$A \rightarrow a$
$B \rightarrow b\}$
This grammar produces strings consisting of a's that are followed by b's. At least there have to be two a's and two b's. Then we obtain the product matrix c by applying the above definition of MM on a and itself:

$$\underbrace{\begin{pmatrix} \emptyset & \{A\} & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \{A\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \{B\} & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \{B\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \end{pmatrix}}_{a} * \underbrace{\begin{pmatrix} \emptyset & \{A\} & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \{A\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \{B\} & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \{B\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \end{pmatrix}}_{a} = \underbrace{\begin{pmatrix} \emptyset & \emptyset & \{X\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \{Y\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \end{pmatrix}}_{c}$$

In the following we observe only matrices with subsets of N as elements. With MM as defined above, the transitive closure of a square matrix can be defined.

Definition 3. *Let a be a $m \times m$ matrix. Then the transitive closure of a, denoted by a^+ is:*

$$a^+ = a^{(1)} \cup a^{(2)} \cup \ldots$$

where

$$a^{(i)} = \bigcup_{j=1}^{i-1} a^{(j)} * a^{(i-j)} \qquad \text{and} \qquad a^{(1)} = a$$

The union of two matrices $a \cup b = c$ is defined as:

$$c_{ij} = a_{ij} \cup b_{ij} \qquad \forall 1 \leq i, j \leq m$$

Note, that computing the transitive closure of a yields a fixpoint and thus the computation is finite.

Example 2. Consider the CFG of example 1. Then the first two iterations of the transitive closure, $a^{(1)} \cup a^{(2)}$, is computed by $a \cup a * a =: c$. Hence we obtain:

$$\begin{pmatrix} \emptyset & \{A\} & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \{A\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \{B\} & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \{B\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \end{pmatrix} \cup \begin{pmatrix} \emptyset & \emptyset & \{X\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \{Y\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \end{pmatrix} = \begin{pmatrix} \emptyset & \{A\} & \{X\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \{A\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \{B\} & \{Y\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \{B\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \end{pmatrix}$$

$$a \qquad \cup \qquad a * a \qquad = \qquad c$$

2.2 Algorithm: CFG Parsing reduced to Matrix Multiplication

Valiant's CFG parsing algorithm requires as input a CFG $G = (N, \Sigma, P, A_1)$ in Chomsky normal form and a string $w = w_1 \dots w_n$ of length n. The string w should be parsed, i.e. the algorithm should check whether w can be derived from the starting symbol A_1.

Now let b be a $(n+1) \times (n+1)$ matrix with subsets of N as elements. The algorithm performs the following steps:

1. $b_{i,j} = \emptyset \qquad \forall 1 \leq i, j \leq (n+1)$
2. $b_{i,i+1} = \{A_k \mid (A_k \to w_i) \in P\} \qquad \forall 1 \leq i \leq n$
3. Compute $a = b^+$
4. Check whether $A_1 \in a_{1,n+1}$

The first step initializes all entries of b with the empty set. Afterwards b is filled by parsing w bottom-up. That means, first all substrings of w of length 1 are parsed (step 2) then all substrings of w of length 2 and so on (step 3). In each iteration of the computation of the transitive closure it holds that:

$$A_k \in b_{ij} \Leftrightarrow A_k \to^* w_i \dots w_{j-1}$$

Hence, if the fixpoint b^+ is reached, it is checked whether the starting symbol A_1 is an element of the set b^+_{1n+1}. If $A_1 \in b^+_{1n+1}$ then A_1 derives $w_1 \dots w_n = w$. Thus w is recognized by G. The following example shows an execution of the algorithm.

Example 3. Let the CFG be defined as in example 1. The string which should be parsed is $w = aabb$. We start at step two of Valiant's algorithm where b is initialized:

2. Build the upper triangular matrix (parsing of all substrings of length 1):

$$b := \begin{pmatrix} \emptyset & \{A\} & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \{A\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \{B\} & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \{B\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \end{pmatrix}$$

3. Compute b^+ :

(a) $a := \underbrace{b^{(1)}}_{=b} \cup \underbrace{b^{(2)}}_{=b*b}$ (parsing of all substrings of length 2)

$$a := \begin{pmatrix} \emptyset & \{A\} & \{X\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \{A\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \{B\} & \{Y\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \{B\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \end{pmatrix}$$

(b) $a := a \cup \underbrace{b^{(3)}}_{=b*b^{(2)} \cup b^{(2)}*b}$ (parsing of all substrings of length 3 and 4)

$$a := \begin{pmatrix} \emptyset & \{A\} & \{X\} & \emptyset & \{S\} \\ \emptyset & \emptyset & \{A\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \{B\} & \{Y\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \{B\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \end{pmatrix}$$

(c) The fixpoint is reached

4. Check whether $S \in a_{1n+1}$: "yes". Hence w can be derived by S and is recognized by the CFG G.

2.3 Time Bounds

The computation of the transitive closure (step 3 of the algorithm) clearly dominates the time complexity of Valiant's CFG parsing algorithm. Hence, the time complexity for computing the transitive closure, denoted by $T(n)$, has to be estimated. It is claimed that Valiant's algorithm is sub-cubic, and thus $T(n)$ has to be sub-cubic.

Theorem 1. *The transitive closure of an upper triangular matrix can be computed in less than cubic time.*

This result is proven in the following. Let $T(n)$ denote the time to compute the transitive closure, $BM(n)$ denotes the complexity for BMM, and with $M(n)$ the complexity for multiplying two matrices is denoted. Figure 2 shows the reduction steps which prove that computing the transitive closure is sub-cubic.

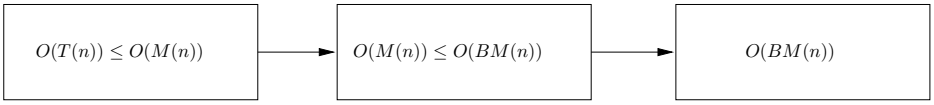

Fig. 2. Time complexity for CFG parsing: $O(BM(n)) = O(n^{2.81})$

First, Valiant shows that computing the transitive closure of an upper triangular matrix has essentially the same time complexity as performing matrix multiplication. Afterwards it is shown that matrix multiplication can be reduced to Boolean matrix multiplication, by simulating matrix multiplication by h^2 Boolean matrix multiplications (where h is the size of the set of nonterminals N).

In the following these two reductions are shown and the involved constants are approximated.

1. Reduction: $O(T(n)) \leq O(M(n))$:
   to show: the transitive closure of an upper triangular matrix b can be computed with the same time complexity as performing matrix multiplication.

   Proof (sketch). Here only a proof sketch is given. For a detailed analysis of the time complexity confer [Val75].
   The proof is inductive over the size of matrix b. Therefore, a recursive procedure for computing the transitive closure is established.
   Matrix b can be partitioned into two smaller upper triangular matrices. It is shown that if the transitive closure of these two matrices is known, b^+ can be computed by performing a single matrix multiplication and computing the transitive closure for a smaller matrix. This leads to recursion and thus, the same time complexity as for matrix multiplication is obtained.

2. Reduction: $O(M(n)) \leq O(BM(n))$:
   to show: we can compute $c = \bigcup_{k=1}^{n} a_{ik} * b_{kj}$ by performing only Boolean matrix multiplications with the same complexity.

   Proof. Let h denote the number of nonterminals, i.e. $h = |N|$. First we form for each nonterminal $A_l \in N$ two $n \times n$ Boolean matrices $a[l]$ and $b[l]$ $(1 \leq l \leq h)$ in the following way:

   $$a[l]_{ik} = 1 \text{ iff } A_l \in a_{ik} \text{ and } b[l]_{kj} = 1 \text{ iff } A_l \in b_{kj} \, \forall 1 \leq i, k, j \leq n.$$

   Hence, we obtain $2h$ Boolean matrices. For each pair l, m with $1 \leq l, m \leq h$ we compute the Boolean matrix $c[l, m]$ by Boolean matrix multiplication where

   $$c[l, m] = a[l] \cdot b[m]$$

   Thus, we obtain h^2 matrices $c[l, m]$. We compute matrix c by

   $$A_p \in c_{ij} \text{ iff } \exists l, m . c[l, m] = 1 \text{ and } (A_p \rightarrow A_l A_m) \in P$$

   Therefore, matrix multiplication can be simulated by h^2 Boolean matrix operations. Since h is a constant we obtain: $O(M(n)) \leq O(BM(n) \cdot h^2) = O(BM(n))$.

By these two reductions it follows that $T(n)$ has the same complexity as performing Boolean matrix multiplication. Since the latter one is proven to be sub-cubic (Algorithm of Strassen: $O(n^{2.81})$), CFG parsing is shown to be sub-cubic. Also it holds that finding a faster BMM algorithm would lead to a faster CFG parsing algorithm.

3 Conversion of a CFG Parser into an Algorithm for Boolean Matrix Multiplication

In the previous section the reduction from CFG parsing to Boolean matrix multiplication was explained. In this section a dual result, i.e. converting a CFG parser into an algorithm for BMM, is shown. This algorithm was developed by Lee in 2002 ([Lee02]).

How this algorithm works in general is depicted in figure 3. As input the BMM

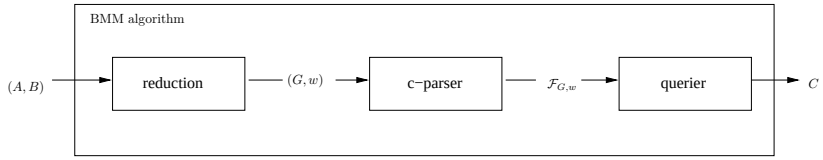

Fig. 3. Conversion of a CFG parser into an algorithm for BMM

algorithm requires two Boolean matrices A and B which should be multiplied. Since a CFG parser should be used, A and B have to be transformed into a CFG G and a string w. w is parsed by a c-parser and the output is an oracle $\mathcal{F}_{(G,w)}$ that answers in constant time whether a nonterminal A_i derives the substring w_j^k. This oracle is passed to the querier which has the task to compute the resulting Boolean matrix C, i.e. the querier answers for all entries of C whether it is 1 or 0. The algorithm with its three components (reduction, c-parser, and querier) is formalized in the following.

3.1 Preliminaries

In the following A and B are Boolean matrices, i.e. matrices with entries 1 and 0. The definition of a CFG G is as usual: G is a 4-tuple (N, Σ, P, S), where N is the set of nonterminals, Σ is the set of terminals, P is the set of productions, and S is the starting symbol. w_i^j denotes a substring of $w = w_1 \dots w_n$ with $w_k \in \Sigma \forall 1 \leq k \leq n$, i.e. $w_i^j = w_i \dots w_j$.

Definition 4. *$A_k \in N$ c-derives w_i^j iff*

(i) $A_k \rightarrow^* w_i^j$, *and*
(ii) $S \rightarrow^* w_1^{i-1} A_k w_{j+1}^n$

C-derivation is stronger than derivation since it requires additionally that the starting symbol S derives the whole string w by using the nonterminal A_k that derives the substring w_i^j. Figure 4 illustrates the definition of c-derivation, i.e. A_k c-derives w_i^j.

With the definition of c-derivation, c-parsers can be defined.

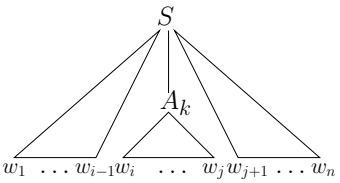

Fig. 4. The nonterminal A_k c-derives w_i^j.

Definition 5. *A c-parser is an algorithm that takes a CFG G and a string w and outputs $\mathcal{F}_{G,w}$ with:*

(i) A_k *c-derives* $w_i^j \Rightarrow \mathcal{F}_{G,w} =$ *"yes"*
(ii) $A_k \not\rightarrow^* w_i^j \Rightarrow \mathcal{F}_{G,w} =$ *"no"*
(iii) $\mathcal{F}_{G,w}$ *answers queries in constant time.*

The input of a c-parser is a CFG G and a string w which should be parsed. The output of a c-parser is an oracle $\mathcal{F}_{G,w}$ that answers in constant time whether a nonterminal c-derives a string w_i^j. To obtain such an oracle, the parser has to generate all possible parse trees for w. Both, CKY and Earley's algorithm, can be used as c-parsers.

Recall the definition for Boolean matrix multiplication.

Definition 6. *The product C of two Boolean matrices A and B is defined as:*

$$A \cdot B = C :\Leftrightarrow c_{ij} = \bigvee_{k=1}^{m} (a_{ik} \wedge b_{kj})$$

From this definition it follows immediately that if $c_{ij} = 1$ then $\exists k.a_{ik} = b_{kj} = 1$.

3.2 Algorithm: BMM reduced to CFG Parsing

As input the algorithm requires two $m \times m$ Boolean matrices A and B. The output should be a Boolean matrix C with $C = A \cdot B$. The next two sections describe the three steps needed to compute the product matrix C using a CFG parser. Confer figure 3 for an overview of these three components.

Reduction In the first step, the reduction step, the two matrices A and B are transformed into a CFG G and a string w that serve as input for the c-parser. Mainly all information about A and B is coded in the grammar. The general idea is to introduce for each entry a_{ij} of A that equals 1 a production $A_{i,j} \rightarrow w_i W w_j$ (A-rule), where W is a nonterminal that can produce an arbitrary string (except the empty string). For all entries b_{ij} of B a production $B_{i,j} \rightarrow w_{i+1} W w_j$ (B-rule) is introduced and finally, for all entries c_{ij} of the resulting matrix C, productions $C_{i,j} \rightarrow A_{i,k} B_{k.j}$ $\forall 1 \leq k \leq m$ (C-rule) are established. In order to check whether c_{ij} has to be set to 1, it has to be answered whether $C_{i,j}$ c-derives w_i^j. Figure 5 shows this graphically. $C_{i,j}$ c-derives $A_{i,k} B_{k,j}$, which c-

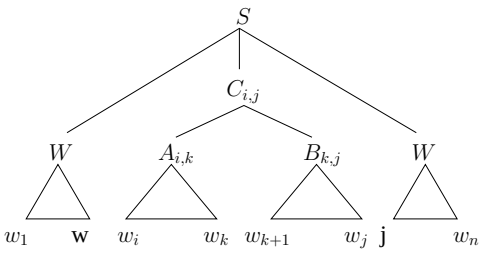

Fig. 5. General idea: Derivation tree for w_1^n where $a_{ik} = b_{kj} = 1$.

derive $w_i W w_k w_{k+1} W w_j = w_i^j$ if and only if there exists a k such that there exist productions for $A_{i,k}$ and $B_{k,j}$. But there only exist such productions if $a_{ik} = b_{kj} = 1$. Hence, c_{ij} is 1 if $C_{i,j}$ c-derives w_i^j.

Unfortunately, the grammar size, which is defined as the sum over all productions, would be very large, i.e. the grammar size would be in $O(m^3)$ since we could have $2m^2$ A- and B-rules, and m^3 C-rules. To keep the grammar size small, and thus also the complexity, all indices like i,j, and k are split into a pair of indices, e.g. $i = (i_1, i_2)$, where i_1 and i_2 are an abbreviation for $f_1(i)$ and $f_2(i)$, respectively. i_1 and i_2 are computed by

$$i_1 = f_1(i) = \lfloor i/d \rfloor \text{ and}$$
$$i_2 = f_2(i) = (i \bmod d) + 2,$$

with $d = \lceil m^{1/3} \rceil$.

It was shown by Lee that the choice of $d = \lceil m^{1/3} \rceil$ results in the best time complexity [Lee02].

Essentially, i_1 and i_2 are the quotient and the remainder of i/d. Thus, i can be computed uniquely by i_1 and i_2. Note that $0 \leq i_1 \leq d^2$ and $2 \leq i_2 \leq d + 1$. i_2 has to be greater than 1 in order to avoid epsilon-rules. Compare figure 6.

Instead of adding productions $A_{i,j} \to w_i W w_j$ productions $A_{i_1,j_1} \to w_{i_2} W w_{j_2}$ are introduced. Analogously for $B_{i,j}$, and $C_{i,j}$. That means the number of productions is smaller since we need only $(d^2)^3 \approx m^2$ C-rules. Thus, the grammar size is in $O(m^2)$. Note that now some information about matrix A is coded in the string w since in order to determine i uniquely i_2 is needed (which is coded in w).

Since i_2 and j_2 are remainders, j_2 could be smaller than i_2. In order to avoid this, a δ that is at least d is added to j_2. Confer figure 6. w consists of three parts x, y, and z, all of size δ. Hence, the substring c-derived by A_{i_1,k_1} should start in x and should end in y. Thus the substring c-derived by B_{k_1,j_1} has to start in y and has to end in z, i.e. A_{i_1,k_1} c-derives to $w_{i_2}^{k_2+\delta}$ and B_{k_1,j_1} c-derives $w_{k_2+1+\delta}^{j_2+2\delta}$. Note that the derived blocks of A_{i_1,k_1} and B_{k_1,j_1} lie directly next to each other. The strings that are derived from the W's should be non-empty in order to avoid epsilon-rules. Therefore the remainder has to be greater than 1 and δ has to be chosen slightly larger than d, i.e. δ is chosen as $d + 2$.

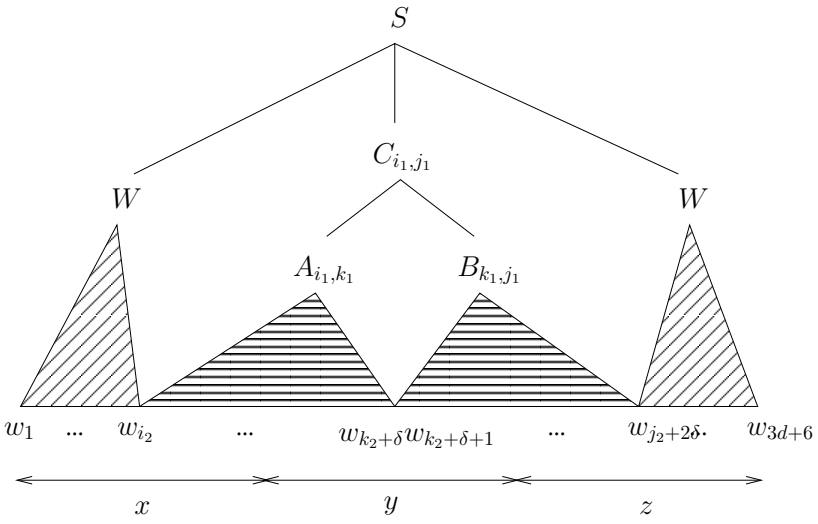

Fig. 6. Derivation tree for w_1^{3d+6} with pairs of indices, where $a_{ik} = b_{kj} = 1$.

Now the resulting CFG and the string w, which should be parsed, for Boolean matrices A and B can be established. Let $G = (N, \Sigma, P, S)$ with $N = \{S\}$ and $P = \emptyset$. Σ is chosen as $\{w_l \mid 1 \leq l \leq 3\delta = 3d + 6\}$ since we need 3δ different terminals in order to compute the indices uniquely. Hence, w is chosen as $w_1 w_2 \dots w_{3d+6}$. The following productions and nonterminals are added to P and N, respectively:

- W-rules: $N = N \cup \{W\}$

$$W \to w_l W \mid w_l \qquad \forall 1 \leq l \leq 3d + 6$$

- A-rules: $N = N \cup \{A_{p,q} \mid 1 \leq p, q \leq d^2\}$
  for all matrix-entries $a_{ij} = 1$ add production

$$A_{i_1,j_1} \to w_{i_2} W w_{j_2+\delta}$$

- B-rules: $N = N \cup \{B_{p,q} \mid 1 \leq p, q \leq d^2\}$
  for all matrix-entries $b_{ij} = 1$ add production

$$B_{i_1,j_1} \to w_{i_2+\delta+1} W w_{j_2+2\delta}$$

- C-rules: $N = N \cup \{C_{p,q} \mid 1 \leq p, q \leq d^2\}$

$$C_{p,q} \to A_{p,r} B_{r,q} \qquad \forall 1 \leq p, q, r \leq d^2$$

- S-rules:

$$S \to W C_{p,q} W \qquad \forall 1 \leq p, q \leq d^2$$

In order to prove that the reduction from Boolean matrices A and B to the CFG G and the string w is valid, the following theorem has to hold.

Theorem 2. *For $1 \leq i, j \leq m$, the entry c_{ij} in C is 1 iff C_{i_1,j_1} c-derives $w_{i_2}^{j_2+2\delta}$.*

Proof. (i) "⇒"

to show: C_{i_1,j_1} c-derives $w_{i_2}^{j_2+2\delta}$

Assume $c_{ij} = 1$. By definition there exists a k such that $a_{ik} = b_{kj} = 1$.

1. Claim: $C_{i_1,j_1} \to^* w_{i_2}^{j_2+2\delta}$:

   $C_{i_1,j_1} \to A_{i_1,k_1} B_{k_1,j_1}$ by C-rule

   $\to^* w_{i_2} W w_{k_2+\delta}\ w_{k_2+\delta+1} W w_{j_2+2\delta}$ $\exists k.a_{ik} = b_{kj} = 1$

   $\to^* w_{i_2}^{j_2+2\delta}$ by W-rule

   Since the W-rules can not produce the empty string we have to show that $w_{i_2}^{k_2+\delta}$ and $w_{k_2+\delta+1}^{j_2+2\delta}$ contain at least one symbol:

   (a) to show: $i_2 + 1 < k_2 + \delta - 1$:
   $i_2 + 1 \leq d + 2 = \delta < 2 + \delta - 1 \leq k_2 + \delta - 1$

   (b) to show: $k_2 + \delta + 2 \leq j_2 + 2\delta - 1$:
   $k_2 + \delta + 2 \leq d + 2 + \delta + 1 = 2\delta + 1 = 2 + 2\delta - 1 \leq j_2 + 2\delta - 1$

   Hence, W derives at least one symbol in the A-rule and in the B-rule.

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

   This follows immediately from the S-rules: $S \to W C_{i_1,j_1} W$ and W can derive all substrings except the empty string. Thus $w_1^{i_2-1}$ and $w_{j_2+2\delta+1}^{3d+6}$ have to consist of at least one symbol:

   (a) to show: $1 \leq i_2 - 1$:
   $i_2 \geq 2 \Rightarrow 1 \leq i_2 - 1$

   (b) to show: $j_2 + 2\delta + 1 \leq 3d + 6$:
   $j_2 + 2\delta + 1 \leq d + 2 + 2\delta = 3\delta = 3d + 6$

   Hence, W derives a non-empty string in the S-rule.

With 1. and 2. it follows that C_{i_1,j_1} c-derives $w_{i_2}^{j_2+2\delta}$.

(ii) "⇐"

to show: $c_{ij} = 1$

Assume C_{i_1,j_1} c-derives $w_{i_2}^{j_2+2\delta}$.

Hence, $C_{i_1,j_1} \to^* w_{i_2}^{j_2+2\delta}$.

$\Rightarrow \exists k_1, k_2. C_{i_1,j_1} \to A_{i_1,k_1} B_{k_1,j_1}$

$\to w_{i_2} W w_{k_2+\delta} w_{k_2+\delta+1} W w_{j_2+2\delta} \to^* w_{i_2}^{j_2+2\delta}$

$\Rightarrow \exists k = (k_1, k_2). a_{ik} = b_{kj} = 1$

$\Rightarrow c_{ij} = 1$

From Theorem 2 follow directly the next two Corollaries.

Corollary 1. *For $1 \leq i, j \leq m$, $c_{ij} = 1$ iff $C_{i_1,j_1} \to^* w_{i_2}^{j_2+2\delta}$. Hence, c-derivation and derivation are equivalent for the $C_{p,q}$ non-terminals.*

This Corollary holds because of the S-rules: $S \rightarrow WC_{p,q}W$. If C_{i_1,j_1} derives $w_{i_2,j_2+2\delta}$ then by the S-rule S derives $W \underbrace{w_{i_2,j_2+2\delta}}_{derived\,by\,C_{i_1,j_1}} W$ which derives w_1^{3d+6}. Hence, C_{i_1,j_1} c-derives $w_{i_2,j_2+2\delta}$.

Corollary 2. *$S \rightarrow^* w$ iff C is not the all-zeroes matrix.*

If C is the all-zeroes matrix there exists no k such that $a_{ik} = b_{kj} = 1\ \forall 1 \leq i,j \leq m$. Hence, there exists no k_1 such that for both nonterminals A_{i_1,k_1} and B_{k_1,j_1} a production exist. Thus, S cannot derive w. Therefore, if a string is accepted by the CFG, C has at least one entry that is 1.

The reduction from Boolean matrices A and B to the CFG G and the string w can now be combined with the c-parser and the querier. This results in an algorithm for BMM using a CFG parser.

C-parser and Querier The shown grammar can be easily converted into Chomsky normal form, since there are no epsilon rules or unit productions. Compare Figure 7. The size of the resulting grammar G' would be also in $O(m^2)$. Therefore, the choice of the c-parser is not restricted, i.e. also a c-parser that requires the grammar to be in Chomsky normal form (e.g. the CKY parser) can be chosen without obtaining a worse complexity.

The c-parser computes all parse trees and outputs an oracle $\mathcal{F}_{(G,w)}$ as it is defined in Definition 5. That means, it can be checked in constant time whether C_{i_1,j_1} c-derives $w_{i_2}^{j_2+2\delta}$. The oracle $\mathcal{F}_{(G,w)}$ is now passed to the querier. The querier answers, by using $\mathcal{F}_{(G,w)}$ and Theorem 2, whether c_{ij} equals 1.

3.3 Time Bounds

The following theorem shows the relation between time bounds for BMM and time bounds for CFG parsing.

Theorem 3. *Any c-parser with running time $O(T(g)t(n))$ can be converted into a BMM algorithm 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 an algorithm for BMM runs in time $O(m^{3-\epsilon/3)}$.*

g denotes the size of the CFG G and n denotes the length of the input string w. m is as usual the size of the input matrices A and B, i.e. A and B are $m \times m$ Boolean matrices.

This theorem says that if there is a CFG parser that is linearly dependent on the grammar size and sub-cubic in the length of the input string, then this CFG parser can be converted into a sub-cubic BMM algorithm, i.e. the BMM algorithm runs in time $O(m^{3-\epsilon/3})$. In order to prove this theorem confer figure 8.

Proof. To read the two input matrices A and B requires $O(m^2)$. Since the size of G is $O(m^2)$ and the size of w is by construction $O(m^{1/3})$, the reduction from (A,B) to (G,w) takes $O(m^2)$. The theorem assumes that the oracle can

– W-rules:

$$W \to W_l W \mid w_l \qquad (1 \le l \le 3d+6)$$
$$W_l \to w_l \qquad (1 \le l \le 3d+6)$$

– A-rules:
for all matrix-entries $a_{ij} = 1$ add productions

$$A_{i_1,j_1} \to W_{i_2} X_{j_2+\delta}$$
$$X_{j_2+\delta} \to W W_{j_2+\delta} \qquad (2 \le j_2 \le d+1)$$

– B-rules:
for all matrix-entries $b_{ij} = 1$ add productions

$$B_{i_1,j_1} \to W_{i_2+1+\delta} X_{j_2+2\delta}$$
$$X_{j_2+2\delta} \to W W_{j_2+2\delta} \qquad (2 \le j_2 \le d+1)$$

– C-rules:

$$C_{p,q} \to A_{p,r} B_{r,q} \qquad (1 \le p,q,r \le d^2)$$

– S-rules:

$$S \to WT$$
$$T \to C_{p,q} W \qquad (1 \le p,q \le d^2)$$

Fig. 7. Grammar of the previous section in Chomsky normal form.

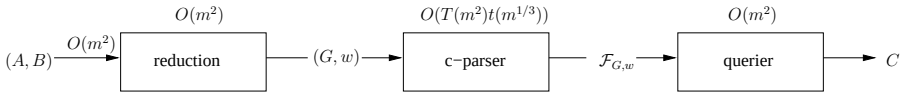

Fig. 8. Time bounds for BMM algorithm using a CFG parser

be computed in $O(T(m^2)t(m^{1/3}))$. Since the oracle $\mathcal{F}_{(G,w)}$ answers queries in constant time, computing the output matrix C takes $O(m^2)$. So the total time spent by the BMM algorithm is $O(max(m^2, T(m^2)t(m^{1/3})))$, as claimed.

In the case where $T(g) = g$ and $t(n) = n^{3-\epsilon}$, the second argument of the maximum in $O(max(m^2, T(m^2)t(m^{1/3})))$ dominates the term. Hence, the running time for the BMM algorithm is $O(T(m^2)t(m^{1/3})) = O(m^2 \cdot (m^{1/3})^{3-\epsilon}) = O(m^{3-\epsilon/3})$.

Since there is a relation between time bounds for BMM and time bounds for CFG parsing, a faster CFG parsing algorithm would lead to a faster BMM algorithm.

4 Conclusion

It was shown, that CFG parsing is possible in less than cubic time. This approach from 1975 is due to Valiant ([Val75]). He shows that CFG parsing can be reduced to BMM obtaining a time complexity of $O(BM(n))$, where $BM(m)$ denotes the complexity to multiply two $m \times m$ matrices.

Afterwards the reverse direction, i.e. how a CFG parser can be used for BMM, was shown. The idea was to transform the input matrices into an input for a c-parser, i.e. a CFG and an input string. After parsing the input, the c-parser outputs an oracle which helps the querier to compute the product matrix. This reduction, which is due to Lee ([Lee02]), was presented and its correctness was proven. Furthermore, by this reduction a relation between the running time of a CFG parser and the running time of an algorithm for BMM was derived, i.e. if a CFG parser has sub-cubic running time in the length of the input string, then a sub-cubic BMM algorithm is obtained.

By reducing BMM to CFG parsing, it was shown that a faster BMM algorithm would yield a faster CFG parsing algorithm. The reverse direction also holds. Thus, the theorem of Lee - "Fast CFG parsing requires fast BMM"(and vice versa) - is proven. The theorem explains, why there is little success in finding a faster CFG parsing algorithm. Since fast practical BMM algorithms are thought not to exist, this establishes a limitation on the efficiency of practical CFG parsers ([Lee02]).

References

[Ear70] Jay Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94-102, 1970.

[Lee02] Lillian Lee. Fast context-free grammar parsing requires fast Boolean matrix multiplication. *Journal of the ACM*, 49(1):1-15, 2002.

[Sat94] Giorgio Satta. Tree-adjoining grammar parsing and Boolean matrix multiplication. *Computational Linguistics*, 20(2):173-191, 1994.

[Str69] Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13:354-356, 1969.

[Val75] Leslie G. Valiant. General context-free recognition in less than cubic time. *Journal of Computer and System Sciences*, 10:308-315, 1975.

[You67] Daniel H. Younger. Recognition and parsing of context-free languages in time n^3. *Information and control*, 10(2):189-208, 1967.