

Contribution Title^{*}

First Author¹[0000–1111–2222–3333], Second Author^{2,3}[1111–2222–3333–4444], and
Third Author³[2222–3333–4444–5555]

¹ Princeton University, Princeton NJ 08544, USA

² Springer Heidelberg, Tiergartenstr. 17, 69121 Heidelberg, Germany
`lncs@springer.com`

<http://www.springer.com/gp/computer-science/lncs>

³ ABC Institute, Rupert-Karls-University Heidelberg, Heidelberg, Germany
`{abc,lncs}@uni-heidelberg.de`

Abstract. The abstract should briefly summarize the contents of the paper in 15–250 words.

Keywords: First keyword · Second keyword · Another keyword.

1 Introduction

Since the theory of context-free grammars was developed by Noam Chomsky, its applications have been studied extensively. Context-free grammars have been used to describe programming languages. But recent research has shown that the theory of formal languages and, in particular, context-free languages can be used in bioinformatics.

It was therefore important to develop more efficient recognition or parsing algorithms. One of the basic parsing algorithms is the CockeKasamiYounger algorithm (CYK) [4, 5], which computes a parsing table with the time complexity $O(n^3)$, where n is the length of the input string.

All further parsing algorithms had the same cubic-time complexity [6] or could work only with sub-classes of context-free grammars [3], until Leslie Valiant presented new, asymptotically more efficient parsing algorithm [1]. Just as the CYK, Valiant's algorithm computes the same parsing table, but its main difference is replacing most of computations with matrix multiplication. Thereafter, Okhotin generalized Valiant's algorithm to the class of boolean grammars and also improved its performance and understandability [2].

This paper aims to present a modification of Valiant's algorithm, which can be simply applied to the string-matching problem, for example, for sequences processing in bioinformatics. Also the algorithm described here is accompanied with the proof of correctness and evaluation of time complexity which is $O(|G|BMM(n)\log(n))$ for an input string of length n , where $BMM(n)$ is the number of operations needed to multiply two Boolean matrices of size $n \times n$.

^{*} Supported by organization x.

2 Background

In this section we briefly describe the key definitions and the basic parsing algorithms which are necessary for further understanding of the results obtained in this paper.

2.1 Preliminaries

Σ is a finite nonempty set called an alphabet, Σ^* is a set of all finite strings over Σ . A grammar is a quadruple (Σ, N, R, S) , where Σ is a finite set of terminals, N is a finite set of nonterminals, R is a finite set of productions of the form $\alpha \rightarrow \gamma$, where $\alpha \in V^*NV^*$, $\gamma \in V^*$, $V = \Sigma \cup N$ and $S \in N$ is a start symbol.

Definition 1. Grammar $G = (\Sigma, N, R, S)$ is called context-free, if $\forall r \in R$ are of the form $A \rightarrow \beta$, where $A \in N, \beta \in V^+$.

Definition 2. Context-free grammar $G = (\Sigma, N, R, S)$ is said to be in Chomsky normal form if $\forall r \in R$ are of the form:

- $A \rightarrow BC$,
- $A \rightarrow a$,
- $S \rightarrow \epsilon$,

where $A, B, C \in N, a \in \Sigma, \epsilon$ is an empty string.

Definition 3. $L_G(A)$ is language of grammar $G_A = (\Sigma, N, R, A)$, which means all the sentences that can be derived in a finite number of steps from the start symbol A.

2.2 Parsing by matrix multiplication

The main problem of parsing is to verify if the input string belongs to the language of some given grammar L_G . In this section we will describe parsing algorithm, based on matrix multiplication. It has been proposed by Leslie Valiant and is the most asymptotically efficient parsing algorithm, which works for all context-free grammars, although they can be generalized to conjunctive and boolean grammars due to Alexander Okhotin.

The CYK algorithm is a basic parsing algorithm. Its main idea is to construct for an input string $a_1a_2...a_n$ a parsing table T of size $n \times n$, where

$$T_{i,j} = \{A | a_{i+1}...a_j \in L_G(A)\} \quad \forall i < j \quad (1)$$

and $G = (\Sigma, N, R, S)$ is a context-free grammar.

The elements of T are filled successively beginning with

$$T_{i-1,i} = \{A | A \rightarrow a_i \in R\} \quad (2)$$

Then,

$$T_{i,j} = f(P_{i,j}) \quad (3)$$

where

$$P_{i,j} = \bigcup_{k=i+1}^{j-1} T_{i,k} \times T_{k,j} \quad (4)$$

$$f(P) = \{A | \exists A \rightarrow BC \in R : (B, C) \in P\} \quad (5)$$

The input string $a_1 a_2 \dots a_n$ belongs to L_G if and only if $S \in T_{0,n}$.

Algorithm 1: Parsing by matrix multiplication: Valiant's Version

Input: Grammar $G = (\Sigma, N, R, S)$, $w = a_1 \dots a_n$, $n \geq 1$, $a_i \in \Sigma$, where $n + 1$ — power of two

```

1  main():
2  compute(0, n + 1);
3  accept if and only if  $S \in T_{0,n}$ 

4  compute( $l, m$ ): if  $m - l \geq 4$  then
5      compute( $l, \frac{l+m}{2}$ );
6      compute( $\frac{l+m}{2}, m$ )
7  end

8  complete( $l, \frac{l+m}{2}, \frac{l+m}{2}, m$ )

9  complete( $l, m, l', m'$ ): if  $m - l = 4$  and  $m = l'$  then
10      $T_{l,l+1} = \{A | A \rightarrow a_{l+1} \in R\}$ ;
11 end
12 else if  $m - l = 1$  and  $m < l'$  then
13      $T_{l,l'} = f(P_{l,l'})$ ;
14 end
15 else if  $m - l > 1$  then
16      $B = (l, \frac{l+m}{2}, \frac{l'+m'}{2}, m')$ ,  $B' = (\frac{l+m}{2}, m, l', \frac{l'+m'}{2})$ ,
         $C = (\frac{l+m}{2}, m, l', \frac{l'+m'}{2})$ ,  $D = (l, \frac{l+m}{2}, l', \frac{l'+m'}{2})$ ,
         $D' = (\frac{l+m}{2}, m, \frac{l'+m'}{2}, m')$ ,  $E = (l, \frac{l+m}{2}, \frac{l'+m'}{2}, m')$ ;
17     complete(C);
18      $P_D = P_D \cup (T_B \times T_C)$ ;
19     complete(D);
20      $P_{D'} = P_{D'} \cup (T_C \times T_{B'})$ ;
21     complete(D');
22      $P_E = P_E \cup (T_B \times T_{D'})$ ;
23      $P_E = P_E \cup (T_D \times T_{B'})$ ;
24     complete(E)
25 end
26 complete( $l, \frac{l+m}{2}, \frac{l+m}{2}, m$ )
```

The time complexity of this algorithm is $O(n^3)$. Valiant proposed to offload the most intensive computations to the Boolean matrix multiplication. As the most time-consuming is computing $\bigcup_{k=i+1}^{j-1} T_{i,k} \times T_{k,j}$, Valiant rearranged computation of $T_{i,j}$, in order to use multiplication of submatrices of T.

Definition 4. Let $X \in (2^N)^{m \times l}$ and $Y \in (2^N)^{l \times n}$ be two submatrices of parsing table T. Then, $X \times Y = Z$, where $Z \in (2^{N \times N})^{m \times n}$ and $Z_{i,j} = \bigcup_{k=1}^l X_{i,k} \times Y_{k,j}$.

In **Algorithm 1** full pseudo-code of Valiant's algorithm written in the terms proposed by Okhotin, is presented. All elements of T and P are initialized by empty sets. Then, the elements of these two table are successively filled by two recursive procedures.

The procedure *compute*(l, m) constructs the correct values of $T_{i,j} \forall l \leq i < j < m$.

The procedure *complete*(l, m, l', m') constructs the submatrix $\forall T_{i,j} \ l \leq i < m, l' \leq j < m'$. This procedure assumes $T_{i,j} \forall l \leq i < j < m, l' \leq i < j < m'$ are already constructed and the current value of $P[i, j] = \{(B, C) | \exists (m \leq k < l') : a_{i+1} \dots a_k \in L(B), a_{k+1} \dots a_j \in L(C)\} \forall l \leq i < m, l' \leq j < m'$.

Then Valiant described that product of multiplying of two submatrices of parsing table T can be provided as $|N|^2$ Boolean matrices (for each pair of non-terminals). Denote matrix corresponding to pair $(B, C) \in N \times N$ as $Z^{(B,C)}$, then $Z_{i,j}^{(B,C)} = 1$ if and only if $(B, C) \in Z_{i,j}$. It should also be noted that $Z^{(B,C)} = X^B \times Y^C$. So, matrix multiplication in **Definition 4** can be replaced by Boolean matrix multiplication, each of which can be computed independently. Following these changes, time complexity of **Algorithm 1** is $O(|G|BMM(n)\log(n))$ for an input string of length n , where $BMM(n)$ is the number of operations needed to multiply two Boolean matrices of size $n \times n$.

3 Modification of Valiant's algorithm

In this section we describe the modification of Valiant's algorithm, which has a number of advantages, such as possibility to broke it down into several subtasks that can be processed independently. Also this version can be simply applied to the string-matching problem, which often arises in text editing, DNA and RNA sequence analysis.

The main change of this modification is to divide the parsing table into layers of disjoint submatrices of the same size. The division, we have made from the reorganization of the matrix multiplication order, is presented in **Figure 1**. The layers are computes successively from the bottom up.

Let us consider the pseudo-code of the modification, which is written in **Algorithm 2**. The procedure *main*() computes the lowest layer ($T_{l,l+1}$), and then divide the table into layers, described earlier, and computes them through the *completeVLayer*() call. Thus, *main*() computes all elements of parsing table T correctly.

Algorithm 2: Parsing by matrix multiplication: Modified Version

Input: Grammar $G = (\Sigma, N, R, S)$, $w = a_1 \dots a_n$, $n \geq 1$, $a_i \in \Sigma$, where $n + 1$ — power of two

```

1  main():
2  for  $l \in \{1, \dots, n\}$  do
3       $T_{l,l+1} = \{A | A \rightarrow a_{l+1} \in R\}$ 
4  end
5  for  $1 \leq i < k$  do
6       $\text{layer} = \text{constructLayer}(i)$ ;
7       $\text{completeVLayer}(\text{layer})$ 
8  end
9  constructLayer(i):
10  $\{B | \exists k \geq 0 : B = (k * 2^i, (k + 1) * 2^i, (k + 1) * 2^i, (k + 2) * 2^i)\}$ 
11 completeLayer(M):
12 if  $\forall (l, m, l', m') \in M \quad (m - l = 1)$  then
13     for  $(l, m, l', m') \in M$  do
14          $T_{l,l'} = f(P_{l,l'})$ ;
15     end
16 end
17 else
18      $\text{bottomLayer} = \{( \frac{l+m}{2}, m, l', \frac{l'+m'}{2} ) | (l, m, l', m') \in M\}$ ;
19      $\text{completeLayer}(\text{bottomLayer})$ ;
20      $\text{completeVLayer}(M)$ 
21 end
22 completeVLayer(M):
23  $\text{leftSubLayer} = \{(l, \frac{l+m}{2}, l', \frac{l'+m'}{2}) | (l, m, l', m') \in M\}$ ;
24  $\text{rightSubLayer} = \{( \frac{l+m}{2}, m, \frac{l'+m'}{2}, m' ) | (l, m, l', m') \in M\}$ ;
25  $\text{topSubLayer} = \{(l, \frac{l+m}{2}, \frac{l'+m'}{2}, m' ) | (l, m, l', m') \in M\}$ ;
26  $\text{multiplicationTask1} =$ 
     $\{(l, m, l', m'), (l, m, m, 2m - l), (m, 2m - l, l', m') | (l, m, l', m') \in \text{leftSubLayer}\} \cup$ 
 $\{(l, m, l', m'), (l, m, 2l' - m', l'), (2l' - m', l', l', m') | (l, m, l', m') \in \text{rightSubLayer}\}$ ;
27  $\text{multiplicationTask2} =$ 
 $\{(l, m, l', m'), (l, m, m, 2m - l'), (m, 2m - l, l', m') | (l, m, l', m') \in \text{topSubLayer}\}$ ;
28  $\text{multiplicationTask3} =$ 
 $\{(l, m, l', m'), (l, m, 2l' - m', l'), (2l' - m', l', l', m') | (l, m, l', m') \in \text{topSubLayer}\}$ ;
29  $\text{performMultiplications}(\text{multiplicationTask1})$ ;
30  $\text{completeLayer}(\text{leftSubLayer} \cup \text{rightSubLayer})$ ;
31  $\text{performMultiplications}(\text{multiplicationTask2})$ ;
32  $\text{performMultiplications}(\text{multiplicationTask3})$ ;
33  $\text{completeLayer}(\text{topSubLayer})$ 

```

Denote some subsidiary functions for matrix m :

- $bottom(m) = (\frac{l+m}{2}, m, l', \frac{l'+m'}{2})$,
- $left(m) = (l, \frac{l+m}{2}, l', \frac{l'+m'}{2})$,
- $right(m) = (\frac{l+m}{2}, m, \frac{l'+m'}{2}, m')$,
- $top(m) = (l, \frac{l+m}{2}, \frac{l'+m'}{2}, m')$.

The procedure $completeVLayer(M)$ takes an array of disjoint submatrices M . For each $m = (l, m, l', m') \in M$ this procedure computes $left(m), right(m), top(m)$. The procedure assumes that the elements of $bottom(m)$ and all $T_{i,j} \forall l \leq i < j < m, l' \leq i < j < m'$ are already constructed. Also it is assumed that the current value of $P[i, j] = \{(B, C) | \exists (m \leq k < l') : a_{i+1} \dots a_k \in L(B), a_{k+1} \dots a_j \in L(C)\} \forall l \leq i < m, l' \leq j < m'$.

The procedure $completeLayer(M)$ also takes an array of disjoint submatrices M , but unlike the previous one, it computes $T_{i,j} \forall (i, j) \in m$. This procedure, just as in the previous case, assumes that $T_{i,j} \forall l \leq i < j < m, l' \leq i < j < m'$ are already constructed and the current value of $P[i, j] = \{(B, C) | \exists (m \leq k < l') : a_{i+1} \dots a_k \in L(B), a_{k+1} \dots a_j \in L(C)\} \forall l \leq i < m, l' \leq j < m'$.

Algorithm 3 describes how the procedure $performMultiplication(task)$, where $task$ is an array of a triple of submatrices, works. It is worth mentioning that, as distinct from the original algorithm, $|tasks| \geq 1$ and all these multiplications can be computed independently.

Algorithm 3:

```

1 performMultiplication(task):
2   for  $(m, m1, m2) \in M$  do
3      $P_m = P_m \cup (T_{m1} \times T_{m2})$ ;
4   end

```

4 Proof of correctness

In this section ...

Theorem 1. *Let M be a submatrix array. Assume that $T[i, j] = \{A | a_{i+1} \dots a_j \in L(A)\} \forall l \leq i < j < m, l' \leq i < j < m'$ and $P[i, j] = \{(B, C) | \exists (m \leq k < l') : a_{i+1} \dots a_k \in L(B), a_{k+1} \dots a_j \in L(C)\} \forall l \leq i < m, l' \leq j < m' \forall (l, m, l', m') \in M$.*

Then the procedure $completeLayer(M)$, returns correctly computed sets of $T[i, j] \forall l \leq i \leq m, l' \leq j \leq m' \forall (l, m, l', m') \in M$.

Proof. Induction on $m - l$. (Hereinafter denoting (l, m, l', m') as a typical example of array M , and all the computations are implemented for all submatrices in M).

The base case: $m - l = 1$. There is only one element to compute, and $P[l, l'] = \{(B, C) | a_{l+1} \dots a_{l'} \in L(B)L(C)\}$. Further, algorithm computes $f(P[l, l']) = \{A | a_{l+1} \dots a_{l'} \in L(A)\}$, so $T[l, l']$ computed correctly.

For the induction step, assume that $(l1, m1, l2, m2)$ is correctly computed for $m2 - l2 = m1 - l1 > m - l$.

Let us consider complete $completeLayer(M)$, where $m - l > 1$.

Firstly, consider $completeLayer(bottom = \{(\frac{l+m}{2}, m, l', \frac{l'+m'}{2})\})$, as theorem conditions are fulfilled, then this call returns correct sets $T[i, j] \forall (i, j) \in bottom$ (hereinafter is means $\forall (i, j) \in m \forall m \in bottom$). All submatrices with size $m1 - l1 > m - l$, all previous layers and also $bottom(M)$ are correct, so, $completeVLayer(M)$ can be called, and $multiplicationByTask(task1)$ adds to each $P[i, j] \forall (i, j) \in left = \{(\frac{l+m}{2}, m, l', \frac{l'+m'}{2})\}$ all pairs $\{(B, C) | \exists (\frac{l+m}{2} \leq k < l') : a_{i+1} \dots a_k \in L(B), a_{k+1} \dots a_j \in L(C)\}$ and $\forall (i, j) \in right = \{(\frac{l+m}{2}, m, \frac{l'+m'}{2}, m')\}$ all pairs $\{(B, C) | \exists (m < k < \frac{l'+m'}{2}) : a_{i+1} \dots a_k \in L(B), a_{k+1} \dots a_j \in L(C)\}$. Now all the theorem conditions are fulfilled so, it is possible to call $completeLayer(left \cup right)$, which returns correct sets $T[i, j] \forall (i, j) \in (left \cup right)$.

Next, $multiplicationByTask(task2)$ and $multiplicationByTask(task3)$ add to each $P[i, j] \forall (i, j) \in top = \{(l, \frac{l+m}{2}, \frac{l'+m'}{2}, m')\}$ all pairs $\{(B, C) | \exists (\frac{l+m}{2} \leq k < m) \text{ and } (l' \leq k < \frac{l'+m'}{2}) : a_{i+1} \dots a_k \in L(B), a_{k+1} \dots a_j \in L(C)\}$. Now all the theorem conditions are fulfilled so, it is possible to call $completeLayer(top)$, which returns correct sets $T[i, j] \forall (i, j) \in top$.

Thus, all $T[i, j] \forall (i, j) \in M$ are computed correctly.

Theorem 2. Let M be a submatrix array. Assume that, $T[i, j] = \{A | a_{i+1} \dots a_j \in L(A)\} \forall l \leq i < j < m, l' \leq i < j < m'$ and $\forall b1 \leq i < b2, b3 \leq j < b4$, where $(b1, b2, b3, b4) = (\frac{l+m}{2}, m, l', \frac{l'+m'}{2})$, also $P[i, j] = \{(B, C) | \exists (m \leq k < l') : a_{i+1} \dots a_k \in L(B), a_{k+1} \dots a_j \in L(C)\} \forall l \leq i < m, l' \leq j < m' \forall (l, m, l', m') \in M$.

Then, the procedure $completeVLayer(M)$, returns correctly computed sets of $T[i, j] \forall l \leq i \leq m, l' \leq j \leq m' \forall (l, m, l', m') \in M$.

Proof. The proof is similar to the proof of Theorem 1.

Note 1. Function $constructLayer(i)$ returns $2^{k-i} - 1$ matrices of size 2^i .

Lemma 1.

- $\forall i \in \{1, \dots, k-1\} \sum |layer|$ for the calls of $completeVLayer(layer)$ where $\forall (l, m, l', m') \in layer$ with $m - l = 2^{k-i}$ is exactly $2^{2i-1} - 2^{i-1}$;
- $\forall i \in \{1, \dots, k-1\}$ products of submatrices of size $2^{k-i} \times 2^{k-i}$ are calculated exactly $2^{2i-1} - 2^i$

Proof. The base case: $i = 1$. $completeVLayer(layer)$ where $\forall (l, m, l', m') \in layer$ with $m - l = 2^{k-1}$ is called only once in the $main()$ and $|layer| = 1$. So, $2^{2i-1} - 2^{i-1} = 2^1 - 2^0 = 1$.

For the induction step, assume that $\forall i \in \{1, \dots, j\} \sum |layer|$ for the calls of $completeVLayer(layer)$ where $\forall (l, m, l', m') \in layer$ with $m - l = 2^{k-i}$ which is exactly $2^{2i-1} - 2^{i-1}$.

Let us consider $i = j + 1$.

Firstly, it is the call of $completeVLayer(costructLayer(k - i))$, where $costructLayer(i)$ returns $2^i - 1$ matrices of size 2^i . Secondly, $completeVLayer(layer)$ is called 3 times for the left, right and top submatrices of size $2^{k-(i-1)}$. Finally, $completeVLayer(layer)$ is called 4 times for the bottom, left, right and top submatrices of size $2^{k-(i-2)}$, except $2^{i-2} - 1$ matrices which were already computed.

Then, $\sum |layer| = 2^i - 1 + 3 \times (2^{2(i-1)-1} - 2^{(i-1)-1}) + 4 \times (2^{2(i-2)-1} - 2^{(i-2)-1}) - (2^{i-2} - 1) = 2^{2i-1} - 2^{i-1}$.

To calculate the number of products of submatrices of size $2^{k-i} \times 2^{k-i}$, we consider the calls of $completeVLayer(layer)$ where $\forall (l, m, l', m') \in layer$ with $m - l = 2^{k-(i-1)}$, which is $2^{2(i-1)-1} - 2^{(i-1)-1}$. During these calls $performMultiplications$ run 3 times, $|multiplicationTask1| = 2 \times 2^{2(i-1)-1} - 2^{(i-1)-1}$ and $|multiplicationTask2| = |multiplicationTask3| = 2^{2(i-1)-1} - 2^{(i-1)-1}$. So, the number of products of submatrices of size $2^{k-i} \times 2^{k-i}$ is $4 \times (2^{2(i-1)-1} - 2^{(i-1)-1}) = 2^{2i-1} - 2^i$.

Theorem 3. *The time complexity of the Algorithm 1 is $O(|G|BMM(n)\log(n))$ for an input string of length n , where G is a context-free grammar in Chomsky normal form, $BMM(n)$ is the number of operations needed to multiply two Boolean matrices of size $n \times n$.*

Proof. The proof is almost identical with that of the theorem given by Okhotin [2], because, as shown in the last lemma, the Algorithm 1 has the same number of products of submatrices.

5 Applications

References

1. Valiant, Leslie G.: General context-free recognition in less than cubic time. Journal of computer and system sciences **10**(2), 308–315 (1975)
2. Okhotin, A.: Parsing by matrix multiplication generalized to Boolean grammars. Theoretical Computer Science **516**, 101–120 (2014)
3. Bernardy, J.P., Claessen K.: Efficient divide-and-conquer parsing of practical context-free languages. In: ACM SIGPLAN Notices, vol. 48. no. 9, pp. 111–122 ACM (2013)
4. Kasami, T.: An efficient recognition and syntax-analysis algorithm for context-free languages. Coordinated Science Laboratory Report no. R-257 (1966)
5. Younger, D.H.: Recognition and parsing of context-free languages in time n^3 . Information and control **10**(2), 189–208 (1967)
6. Earley, J.: An efficient context-free parsing algorithm. Communications of the ACM **13**(2), 94–102 (1970)