On the Valiant's Algorithm Parallelization[⋆]

Yuliya Susanina[1,2], Anna Yaveyn[1], and Semyon Grigorev[1,2][0000−0002−7966−0698]

[1] Saint Petersburg State University, 7/9 Universitetskaya nab.
St. Petersburg, 199034 Russia
[2] JetBrains Research, Universitetskaya emb., 7-9-11/5A
St.Petersburg, Russia
jsusanina@gmail.com, anya.yaveyn@yandex.ru, semen.grigorev@jetbrains.com

Abstract. Recent research has shown that the theory of formal languages can be used in bioinformatics. While using parsing for processing nucleotide sequences it became necessary to find an easily adaptable to the string-matching problem parsing algorithm, and as such field of application as bioinformatics requires working with large amount of data, it should be highly efficient. The asymptotically fastest parsing algorithm was proposed by Valiant and based on matrix multiplication. The original version of matrix-based algorithm is difficult to apply to the string-matching problem. In this paper we present a modification of Valiant's algorithm dealing with the problem mentioned above. The modification has a succinct proof of correctness and is implemented.

Keywords: Parsing · Matrix multiplication · Context-free grammars.

1 Introduction

Since the theory of context-free grammars was developed by Noam Chomsky, it has been extensively studied [3, 4]. The classic application of context-free grammars is describing natural and programming languages. Recent research has shown that the theory of formal languages and, in particular, context-free languages can be used in bioinformatics [**?**,**?**,**?**,**?**].

A good example of this usage is the recognition and classification problems in bioinformatics, some of them are based on the research claiming that the secondary structure of the DNA and RNA nucleotide sequence contains an important information about the organism species. The specific features of the secondary structure can be described by some context-free grammar, and therefore the recognition problem can be reduced to parsing—verification if some nucleotide sequence can be derived in this grammar.

Such field of application as bioinformatics requires working with large amount of data, so it is necessary to find highly efficient parsing algorithm. Moreover, this algorithm needs to be easily adaptive to such computing techniques as GPGPU (General-Purpose computing on Graphics Processing Units) or CPU parallel

⋆ The research was supported by the Russian Science Foundation, grant No. 18-11-00100.

computing which is now a fairly widespread method to accelerate the computation.

The majority of parsing algorithms either has the cubic-time complexity (Kasami [6], Younger [9], Earley [5]) or could work only with sub-classes of context-free grammars (Bernardy, Claussen [2]), but still asymptotically more efficient parsing algorithm that can be applied to any context-free grammar is algorithm based on matrix multiplication proposed by Leslie Valiant [8]. It computes the parsing table for a linear input, where each element of this table is responsible for deriving a particular substring. By offloading the most time-consuming computations on a Boolean matrix multiplication, Valiant has achieved an improvement in time complexity, which is O(BMM(n)log(n)) for an input string of size n, where BMM(n) is the number of operations needed to multiply two Boolean matrices of size n × n. In addition, Okhotin generalized the matrix-based algorithm to conjunctive and Boolean grammars which are the natural extensions of context-free grammars with more expressive power and also improved its performance and understandability [7]. In spite of the fact that Valiant's algorithm allows us to use parallel techniques, for example, compute matrix products on GPUs, it seems like a large part of matrix multiplications can be performed concurrently.

In this paper we show how to reorganize the matrix multiplication order in Valiant's algorithm to divide the parsing table into successively computed layers of disjoint submatrices where each submatrix of the layer can be processed independently.

We make the following contributions:

- We propose the modification of Valiant's algorithm which allows to compute some matrix products concurrently and improve the performance through parallel techniques.
- We prove the correctness of the modification and provide its time complexity estimation 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 $n \times n$.
- We show the applicability of our approach in bioinformatics research, especially in addressing the string-matching problem.
- We have implemented the proposed algorithm and our evaluation shows that ... parallel techniques improve the performance...

2 Background

In this section we briefly introduce the key definitions and the necessary parsing algorithm. Before descibing the Valiant's algorithm, we would like to mention one of the basic recognition algorithms known as CYK (the Cocke-Younger-Kasami algorithm), by which we show the main Valiant's idea that made such time complexity possible.

2.1 Preliminaries

An alphabet Σ is a finite nonempty set of symbols. 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 $\alpha \to \gamma$, 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 $A \to \beta$, 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 all productions in R are of the form:*

- *$A \to BC$,*
- *$A \to a$,*
- *$S \to \varepsilon$,*

where $A, B, C \in N, a \in \Sigma, \varepsilon$ 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 rules applications 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 G. We will describe the Cocke-Younger-Kasami algorithm and the most asymptotically efficient parsing algorithm, which works for all context-free grammars, Valiant's parsing algorithm, based on matrix multiplication. In this paper we use the rewritten version of Valiant's algorithm proposed by Alexander Okhotin.

The CYK algorithm is a basic parsing algorithm. Its main idea is to construct a parsing table T of size $n \times n$ for an input string $a_1a_2 \dots a_n$ and context-free grammar $G = (\Sigma, N, R, S)$ which is in Chomsky normal form, where

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

The elements of T are filled successively beginning with $T_{i-1,i} = \{A | A \to a_i \in R\}$. Then, $T_{i,j} = f(P_{i,j})$ where

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

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

The input string $a_1a_2 \dots a_n$ belongs to $L_G(S)$ if and only if $S \in T_{0,n}$.

Listing 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 is a
       power of two
main():
compute(0, n + 1);
accept if and only if $S \in T_{0,n}$

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

complete(l, m, l', m'):
if $m - l = 4$ and $m = l'$ then
    $T_{l,l+1} = \{A | A \to a_{l+1} \in R\}$;
else if $m - l = 1$ and $m < l'$ then
    $T_{l,l'} = f(P_{l,l'})$;
else if $m - l > 1$ then
    $leftgrounded = (l, \frac{l+m}{2}, \frac{l+m}{2}, m), rightgrounded = (l', \frac{l'+m'}{2}, \frac{l'+m'}{2}, m')$,
    $bottom = (\frac{l+m}{2}, m, l', \frac{l'+m'}{2}), left = (l, \frac{l+m}{2}, l', \frac{l'+m'}{2})$,
    $right = (\frac{l+m}{2}, m, \frac{l'+m'}{2}, m'), top = (l, \frac{l+m}{2}, \frac{l'+m'}{2}, m')$;
    complete(bottom);
    $P_{left} = P_{left} \cup (T_{leftgrounded} \times T_{bottom})$;
    complete(left);
    $P_{right} = P_{right} \cup (T_{bottom} \times T_{rightgrounded})$;
    complete(right);
    $P_{top} = P_{top} \cup (T_{leftgrounded} \times T_{right})$;
    $P_{top} = P_{top} \cup (T_{left} \times T_{rightgrounded})$;
    complete(top)
```

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 listing 1 full pseudo-code of Valiant's algorithm is 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.

In figure 1 is presented a simple example of Valiant's algorithm. Only the beginning of the work is shown, because later we point out at this version and our approach differences.

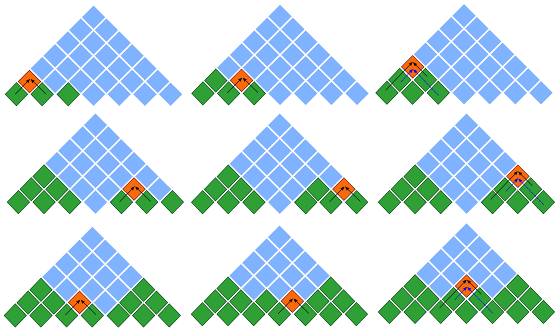
Fig. 1: An example of beginning of Valiant's algorithm

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 $P[i, j] = \{(B, C) | \exists (m \leq k < l') : a_{i+1} \dots a_k \in L(B), a_{k+1} \dots a_j \in L(C)\}\ \forall l \leq i < m, l' \leq j < m'$. The submatrix division during the procedure call is shown in figure 2.

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 $Z^{(B,C)}_{i,j} = 1$ 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 in listing 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 $n \times n$.

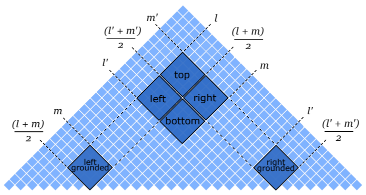

Fig. 2: Matrix partition used in *complete(l, m, l', m')* procedure.

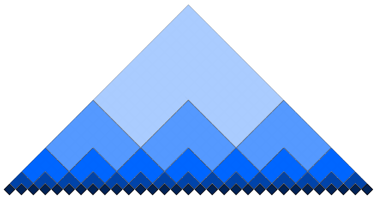
Fig. 3: Matrix partition on V-shaped layers.

3 Modified Valiant's algorithm

In this section we describe the reorganization of submatrix processing order in the Valiant's algorithm which simplify independent handling of submatrices. As a result, proposed modification can facilitate implementation of parallel submatrix processing.

3.1 New approach

The main change of this modification is the possibility to divide the parsing table into layers of disjoint submatrices of the same size. The idea of division we have made from the reorganization of the matrix multiplication order is presented in figure 3. Each layer consists of square matrices which size is power of 2. The layers are computed successively in the bottom-up order. Each matrix in the layer can be handled independently, which can help to implement parallel version of layer processing function.

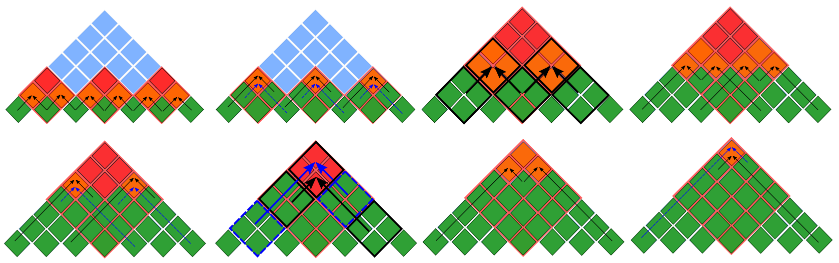
Fig. 4: An example of the modification of Valiant's algorithm

A simple example of the modification is shown in figure 4. The lowest layer (submatrices which size is 1) is already computed and filling of the matrix starts with the second layer (????subfigures???? 1-2). Note that the same process is presented in figure 1, but here it can be done only in two steps using parallel computation of submatrix products.

The modified version of Valiant's algorithm is presented in listing 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. (Hereinafter, we use layer to mean set of submatrices.)

For brevity, we define *left(subm), right(subm), top(subm), bottom(subm), rightgrounded(subm)* and *leftgrounded(subm)* functions which returns the submatrices for matrix $subm = (l, m, l', m')$ according to the original Valiant's algorithm (figure 2).

Also denote some subsidiary functions for matrix layer M:

– $bottomsublayer(M) = \{bottom(subm) | subm \in M\}$,

– *leftsublayer(M)* $= \{left(subm) | subm \in M\}$,
– *rightsublayer(M)* $= \{right(subm) | subm \in M\}$,
– *topsublayer(M)* $= \{top(subm) | subm \in M\}$.

Listing 2: Parsing by matrix multiplication: Modified Version

```
Input: $G = (\Sigma, N, R, S), w = a_1 \dots a_n, n \geq 1, n+1 = 2^p, a_i \in \Sigma$
main():
for $l \in \{1, \dots, n\}$ do
    $T_{l,l+1} = \{A | A \to a_{l+1} \in R\}$
for $1 \leq i < k$ do
    layer = constructLayer(i);
    completeVLayer(layer)
constructLayer(i):
$\{B | \exists k \geq 0 : B = (k * 2^i, (k+1) * 2^i, (k+1) * 2^i, (k+2) * 2^i)\}$
completeLayer(M):
if $\forall (l, m, l', m') \in M \quad (m - l = 1)$ then
    for $(l, m, l', m') \in M$ do
        $T_{l,l'} = f(P_{l,l'})$;
else
    completeLayer(bottomsublayer(M));
    completeVLayer(M)
comleteVLayer(M):
$multiplicationTasks_1 = \{left(m_i),\ leftgrounded(m_i),\ bottom(m_i) | m_i \in M\} \cup$
    $\{right(subm), bottom(subm), rightgrounded(subm) | subm \in M\}$;
multiplicationTask$_2$ = $\{top(subm), leftgrounded(subm), right(subm) | subm \in M\}$;
multiplicationTask$_3$ = $\{top(subm), left(subm), rightgrounded | subm \in M\}$;
performMultiplications(multiplicationTask$_1$);
completeLayer(leftsublayer(M) $\cup$ rightsublayer(M));
performMultiplications(multiplicationTask$_2$);
performMultiplications(multiplicationTask$_3$);
completeLayer(topsublayer(M))
performMultiplication(tasks):
for $(m, m1, m2) \in tasks$ do
    $P_m = P_m \cup (T_{m1} \times T_{m2})$;
```

The procedure *completeVLayer(M)* takes an array of disjoint submatrices M which represents a layer. For each *subm = (l, m, l', m')* $\in M$ this procedure computes *left(subm), right(subm), top(subm)*. The procedure assumes that the elements of *bottom(subm)* and $T_{i,j}$ for all i and j such that $l \leq i < j < m$ and $l' \leq i < j < m'$ are already constructed. Also it is assumed that the current value of $P_{i,j} = \{(B,C) | \exists k, (m \leq k < l'), a_{i+1} \dots a_k \in L_G(B), a_{k+1} \dots a_j \in L_G(C)\}$ for all i and j such that $l \leq i < m$ and $l' \leq j < m'$.

The procedure *completeLayer(M)* also takes an array of disjoint submatrices M, but unlike the previous one, it computes $T_{i,j}$ for all $(i,j) \in subm$. This procedure requires exactly same assumptions on $T_{i,j}$ and $P_{i,j}$ as in the previous case.

In the other words, *completeVLayer(M)* computes the entire layer M and *completeLayer(M$_2$)* is a support function which is necessary for computation of smaller square submatrices $subm_2 \in M_2$ inside of M.

Finally, the procedure *performMultiplication(tasks)*, where *tasks* is an array of a triple of submatrices, perform basic step of algorithm: matrix multiplication. It is worth mentioning that, as distinct from the original algorithm, here $|tasks| \geq 1$ and each task can be computed independently. So, practical implementation of this procedure can easily involve different techniques of parallel array processing, such as OpenMP **??**.

3.2 Correctness and complexity

We provide the proof of correctness and time complexity for the proposed modification in this section. To do it we should prove correctness of subprocedure *completeLayer*.

Theorem 1. *Let M be a layer. If for all $(l,m,l',m') \in M$:*

1. $T_{i,j} = \{A | a_{i+1} \dots a_j \in L_G(A)\}$ *for all i and j such that $l \leq i < j < m$ and $l' \leq i < j < m'$;*
2. $P_{i,j} = \{(B,C) | \exists k, (m \leq k < l') : a_{i+1} \dots a_k \in L_G(B), a_{k+1} \dots a_j \in L_G(C)\}$ *for all $l \leq i < m$ and $l' \leq j < m'$.*

Then the procedure completeLayer(M), returns correctly computed sets of $T_{i,j}$ for all $l \leq i \leq m$ and $l' \leq j \leq m'$ for all $(l,m,l',m') \in M$.

Proof. (Proof by induction on $m - l$.)

Let (l,m,l',m') is a typical element of array M. As far as each element can be handled independently, we prove statements only for one element of M.

Basis: $m - l = 1$. 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.

Inductive step: Assume that (l_1, m_1, l_2, m_2) is correctly computed for $m_2 - l_2 = m_1 - l_1 < m - l$.

Consider *completeLayer(M)* call, where $m - l > 1$.

Firstly, consider *completeLayer(bottomsublayer(M))*. Theorem conditions are fulfilled, then this call returns correct sets $T_{i,j}$ for all $(i,j) \in bottomsublayer(M)$. It should be noted that the theorem conditions allow $T_{m,l'}$ (the lowest element) be correctly computed as in base case.

As *bottomsublayer(M)* now calculated,*completeVLayer(M)* can be called.

All $T_{i,j}$ are already calculated for all i and j such that $l \leq i < j < m$ and $l' \leq i < j < m'$, because of the theorem conditions.

Firstly, *performMultiplications(multiplicationTask$_1$)* adds to each $P_{i,j}$ all pairs (B,C) such that $\exists k$, $(\frac{l+m}{2} \leq k < l')$, $a_{i+1} \dots a_k \in L_G(B)$, $a_{k+1} \dots a_j \in L_G(C)$ for all $(i,j) \in leftsublayer(M)$ and (B,C) such that $\exists k$, $(m \leq k < \frac{l'+m'}{2})$, $a_{i+1} \dots a_k \in L_G(B)$, $a_{k+1} \dots a_j \in L_G(C)$ for all $(i,j) \in rightsublayer(M)$. Now *completeLayer(leftsublayer(M) ∪ rightsublayer(M))* can be called and it returns correctly computed *leftsublayer(M) ∪ rightsublayer(M)*.

Then *performMultiplications* called with arguments *multiplicationTask$_2$* and *multiplicationTask$_3$* adds pairs (B,C) such that $\exists k$, $(\frac{l+m}{2} \leq k < m)$, $a_{i+1} \dots a_k \in L_G(B)$, $a_{k+1} \dots a_j \in L_G(C)$ and (B,C) such that $\exists k$, $(l' \leq k < \frac{l'+m'}{2})$, $a_{i+1} \dots a_k \in L_G(B)$, $a_{k+1} \dots a_j \in L_G(C)$ to each $P_{i,j}$ for all $(i,j) \in topsublayer(M)$. So as $m = l'$ (from the construction of the layer), condition for elements of matrix P are fulfilled. Now *completeLayer(topsublayer(M))* can be called and it returns correctly computed *topsublayer(M)*.

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

Theorem 2. *Algorithm from listing 2 correctly computes $T_{i,j}$ for all i and j, thus an input string $a = a_1 a_2 \dots a_n \in L_G(S)$ if and only if $S \in T_{0,n}$.*

Proof. Primarily to prove the theorem, we show by induction that all layers of the parsing table T are computed correctly.

Basis: layer of size 1×1. Parsing table T consists of one layer of size 1 and its elements are correctly computed in lines 2-3 in listing 2.

Inductive step: assume any layer of size less than or equal to $2^{p-2} \times 2^{p-2}$ are computed correctly.

Define layer of size $2^{p-1} \times 2^{p-1}$ as M. Hereinafter *subm = (l, m, l', m')* is a typical element of layer M.

Consider *completeVLayer(M)* call.

All $T_{i,j}$ are already calculated for all i and j such that $l \leq i < j < m$ and $l' \leq i < j < m'$, because these elements lie in layers which are already computed.

Firstly, *performMultiplications(multiplicationTask$_1$)* adds to each $P_{i,j}$ all pairs (B,C) such that $\exists k$, $(\frac{l+m}{2} \leq k < l')$, $a_{i+1} \dots a_k \in L_G(B)$, $a_{k+1} \dots a_j \in L_G(C)$ for all $(i,j) \in leftsublayer(M)$ and (B,C) such that $\exists k$, $(m \leq k < \frac{l'+m'}{2})$, $a_{i+1} \dots a_k \in L_G(B)$, $a_{k+1} \dots a_j \in L_G(C)$ for all $(i,j) \in rightsublayer(M)$. Now *completeLayer(leftsublayer(M) ∪ rightsublayer(M))* can be called and it returns correctly computed *leftsublayer(M) ∪ rightsublayer(M)*.

Then *performMultiplications* called with arguments *multiplicationTask$_2$* and *multiplicationTask$_3$* adds pairs (B,C) such that $\exists k$, $(\frac{l+m}{2} \leq k < m)$, $a_{i+1} \dots a_k \in L_G(B)$, $a_{k+1} \dots a_j \in L_G(C)$ and (B,C) such that $\exists k$, $(l' \leq k < \frac{l'+m'}{2})$, $a_{i+1} \dots a_k \in L_G(B)$, $a_{k+1} \dots a_j \in L_G(C)$ to each $P_{i,j}$ for all $(i,j) \in topsublayer(M)$. So as $m = l'$ (from the construction of the layer), condition for elements of matrix P are fulfilled. Now *completeLayer(topsublayer(M))* can be called and it returns correctly computed *topsublayer(M)*.

Thus, *completeVLayer(M)* returns correct $T_{i,j}$ for all $(i,j) \in M$ for any layer M of parsing table T and lines 4-6 in listing 2 return all $T_{i,j} = \{A | A \in N, a_{i+1} \dots a_j \in L_G(A)\}$.

Lemma 1. *Let $calls_i$ is a number of the calls of completeVLayer(M) where for all $(l,m,l',m') \in M$ with $m - l = 2^{k-i}$.*

- *for all $i \in \{1, .., k-1\}$ $\sum_{n=1}^{calls_i} |M|$ is exactly $2^{2i-1} - 2^{i-1}$;*
- *for all $i \in \{1, .., k-1\}$ products of submatrices of size $2^{k-i} \times 2^{k-i}$ are calculated exactly $2^{2i-1} - 2^i$ times.*

Proof. Prove the first statement by induction on i.

Basis: i = 1. $calls_1$ and $|M| = 1$. So, $2^{2i-1} - 2^{i-1} = 2^1 - 2^0 = 1$.

Inductive step: assume that $\sum_{n=1}^{calls_i} |M|$ is exactly $2^{2i-1} - 2^{i-1}$ for all $i \in \{1, .., j\}$.

Let us consider $i = j + 1$.

Firstly, note that function *costructLayer(i)* returns $2^{k-i} - 1$ matrices of size 2^i, so in the call of *completeVLayer(costructLayer(k - i)) costructLayer(k - i)* returns $2^i - 1$ matrices of size 2^{k-i}. Secondly, *completeVLayer(M)* is called 3 times for the left, right and top submatrices of size $2^{k-(i-1)}$. Finally, *completeVLayer(M)* 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, $\sum_{n=1}^{calls_i} |M| = 2^i - 1 + 3 \times (2^{2(i-1)-1} - 2^{(i-1)-1}) + 4 \times (2^{2(i-2)-1} - 2^{(i-2)-1}) - (2^{i-2} - 1) = 2^{2i-1} - 2^{i-1}$.

Now we know that $\sum_{n=1}^{calls_{i-1}} |M|$ is $2^{2(i-1)-1} - 2^{(i-1)-1}$ and we can calculate the number of products of submatrices of size $2^{k-i} \times 2^{k-i}$. 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. *Let $|G|$ be a length of the description of the grammar G and let n be a length of an input string. Then algorithm from listing 2 calculates matrix T in $\mathcal{O}(BMM(n) \log n)$ where 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 [7], because, as shown in the last lemma, the Algorithm 1 has the same number of products of submatrices.

To summarize, the correctness of the modification was proved and it was shown that the time complexity remained the same as in Valiant's version.

3.3 Algorithm for substrings

Next we show how our modification can be applied to the string-matching problem.

So if we want to find all substrings of size s which can be derived from a start symbol for an input string of size $n = 2^k$, we need to compute layers with submatrices of size not greater than $2^{l'}$, where $2^{l'-2} < s \leq 2^{l'-1}$.

Let $l' = k - (m - 2)$ and consequently $(m - 2) = k - l'$.

For any $m \leq i \leq k$ products of submatrices of size 2^{k-i} are calculated exactly $2^{2i-1} - 2^i$ times and each of them imply multiplying $\mathcal{O}(|G|)$ Boolean submatrices.

$$C \sum_{i=m}^{k} 2^{2i-1} \cdot 2^{\omega(k-i)} \cdot f(2^{k-i}) = C \cdot 2^{\omega l'} \sum_{i=2}^{l'} 2^{(2-\omega)i} \cdot 2^{2(k-l')-1} \cdot f(2^{l'-i}) \leq$$

$$C \cdot 2^{\omega l'} f(2^{l'}) \cdot 2^{2(k-l')-1} \sum_{i=2}^{l'} 2^{(2-\omega)i} = BMM(2^{l'}) \cdot 2^{2(k-l')-1} \sum_{i=2}^{l'} 2^{(2-\omega)i}$$

Thus, time complexity for searching all substrings is $O(|G|BMM(2^{l'})(l'-1))$, while time complexity for the full input string is $O(|G|BMM(2^k)(k-1))$. In contract to the modification, Valiant's algorithm completely calculate at least 2 triangle submatrices of size $\frac{n}{2}$ (as shown in figure 5) which mean minimum asymptotic complexity $O(|G|BMM(2^{k-1})(k-2))$. Make a conclusion that the modification is asymptotically faster for substrings of size $s \ll n$ than the original algorithm.

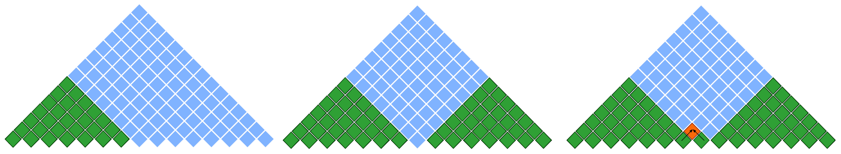
Fig. 5: The number of elements necessary to compute in Valiant's algorithm. That means it is nessesary to calculate at least 2 triangle submatrices of size $\frac{n}{2}$.

4 Evaluation

After demonstrating the theoretical value of the proposed solution, the next step of our work was to show its applicability on real data. Both algorithms (original Valiant's version and the modification) were compared on context-free grammar G ...?picture?... which is used to approximate the secondary structure of the biological sequences. As it was mentioned before, the secondary structure is a very powerful instrument for species classification and identification problem. Parsing algorithms based on matrix multiplication helps efficiently find subsequences with features specific to the secondary structure.

The algorithms were implemented using a library for fast Boolean matrix multiplication M4RI [1]. The biological sequences were taken from this dataset[].

All tests were run on a PC with the following characteristics:

- OS:
- CPU:

– System Type:
– RAM:

The results of experiments which are presented ?...? show that our modification can be efficiently applied to the string matching problem as it demonstrates good time on real data.

5 Conclusion and Future Work

The main goal of this work was to find an effective solution for the string-matching problem that arose at the intersection of the theory of formal languages and bioinformatics. We proposed a modification of Valiant's algorithm partially dealing with this problem. Also we proved its applicability by showing the asymptotic complexity concerning substring searching.

References

1. Albrecht, M., Bard, G.: The M4RI Library. The M4RI Team (2019), https://bitbucket.org/malb/m4ri
2. Bernardy, J.P., Claessen, K.: Efficient divide-and-conquer parsing of practical context-free languages. SIGPLAN Not. **48**(9), 111–122 (Sep 2013), http://doi.acm.org/10.1145/2544174.2500576
3. Chomsky, N.: On certain formal properties of grammars. Information and control **2**(2), 137–167 (1959)
4. Chomsky, N., Schützenberger, M.P.: The algebraic theory of context-free languages. In: Studies in Logic and the Foundations of Mathematics, vol. 35, pp. 118–161. Elsevier (1963)
5. Earley, J.: An efficient context-free parsing algorithm. Commun. ACM **13**(2), 94–102 (Feb 1970), http://doi.acm.org/10.1145/362007.362035
6. Kasami, T.: An efficient recognition and syntax analysis algorithm for context-free languages. Tech. Rep. AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford, MA† (1965)
7. Okhotin, A.: Parsing by matrix multiplication generalized to boolean grammars. Theor. Comput. Sci. **516**, 101–120 (Jan 2014), http://dx.doi.org/10.1016/j.tcs.2013.09.011
8. Valiant, L.G.: General context-free recognition in less than cubic time. J. Comput. Syst. Sci. **10**(2), 308–315 (Apr 1975), http://dx.doi.org/10.1016/S0022-0000(75)80046-8
9. Younger, D.H.: Context-free language processing in time n3. In: Proceedings of the 7th Annual Symposium on Switching and Automata Theory (Swat 1966). pp. 7–20. SWAT '66, IEEE Computer Society, Washington, DC, USA (1966), https://doi.org/10.1109/SWAT.1966.7