Parsing by Matrix Multiplication for String Matching *

Yuliya Susanina$^{1,2[0000-1111-2222-3333]}$, Anna Yaveyn$^{3[1111-2222-3333-4444]}$, and Semyon Grigorev$^{1,2[2222--3333-4444-5555]}$

[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, semen.grigorev@jetbrains.com
[3] ...
. . .

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 · String-matching · Bioinformatics.

1 Introduction

Since the theory of context-free grammars was developed by Noam Chomsky, it has been extensively studied [1, 2]. 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 [3–6].

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.

* Supported by organization x.

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

The majority of parsing algorithms either has the cubic-time complexity (Kasami [8], Younger [9], Earley [10]) or could work only with sub-classes of context-free grammars (Bernardy, Claussen [7]), 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 [11]. 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 [12]. 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 $\forall r \in R$ are of the form:

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

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 rule 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 for an input string $a_1a_2...a_n$ a parsing table T of size $n \times n$, where

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

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

The elements of T are filled successively beginning with

$$T_{i-1,i} = \{A | A \to a_i \in R\} \tag{2}$$

Then,

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

where

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

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

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

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

```
Input: Grammar $G = (\Sigma, N, R, S), w = a_1 ... a_n, n \geq 1, a_i \in \Sigma$, where n + 1 — 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 $\bigcup\limits_{k=i+1}^{j-1} T_{i,k} \times T_{k,j}$, 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 $Z_{i,j} = \bigcup\limits_{k=1}^{l} X_{i,k} \times Y_{k,j}$.

In **Algorithm 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.

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

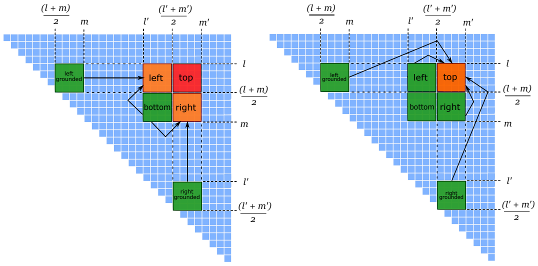

Fig. 1. Matrix partition used in *complete*(l, m, l', m') procedure

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 nonterminals). 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 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.

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 division we have made from the reorganization of the matrix multiplication order is presented in **Figure 2**. The layers are computed successively from the bottom up.

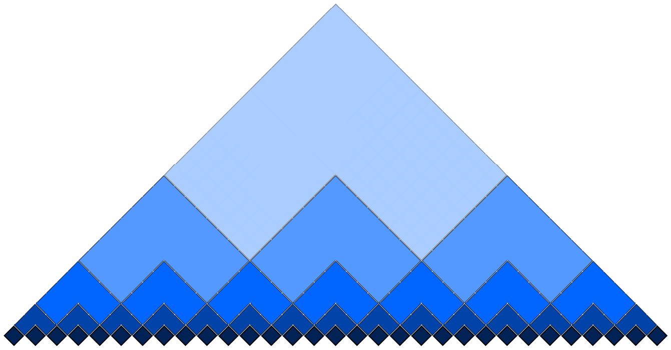

Fig. 2. Matrix partition on V-shaped layers.

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.

For the sake of brevity, we introduce $left(m), right(m), top(m), bottom(m)$, $rightgrounded(m)$ and $leftgrounded(m)$ functions which returns the nessesary submatrix for matrix $m = (l, m, l', m')$ according to **Figure 1**.

Denote some subsidiary functions for matrix layer M:

- $bottomsublayer(M) = \{bottom(m) | m \in M\}$,
- $lrsublayer(M) = \{left(m) | m \in M\} \cup \{right(m) | m \in M\}$,
- $topsublayer(M) = \{top(m) | m \in M\}$.

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 $P[i,j] = \{(B,C) | \exists (m \leq k < l') : a_{i+1}...a_k \in L(B), a_{k+1}...a_j \in L(C)\} \ \forall l \leq i < m, l' \leq j < m'$.

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 $P[i,j] = \{(B,C) | \exists (m \leq k < l') : a_{i+1}...a_k \in L(B), a_{k+1}...a_j \in L(C)\} \ \forall l \leq i < m, l' \leq j < m'$.

Algorithm 2: Parsing by matrix multiplication: Modified Version

```
Input: Grammar $G = (\Sigma, N, R, S), w = a_1...a_n, n \geq 1, a_i \in \Sigma$, where n + 1 — power
       of two
main():
for $l \in \{1, \ldots, n\}$ 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):
multiplicationTask1 =
  $\{left(subm), leftgrounded(subm), bottom(subm) | subm \in M\} \cup$
  $\{right(subm), bottom(subm), rightgrounded(subm) | subm \in M\}$;
multiplicationTask2 = $\{top(subm), leftgrounded(subm), right(subm) | subm \in M\}$;
multiplicationTask3 = $\{top(subm), left(subm), rightgrounded | subm \in M\}$;
performMultiplications(multiplicationTask1);
completeLayer(lrsublayer(M));
performMultiplications(multiplicationTask2);
performMultiplications(multiplicationTask3);
completeLayer(topsublayer(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:

```
performMultiplication(task):
for $(m, m1, m2) \in M$ do
    $P_m = P_m \cup (T_{m1} \times T_{m2})$;
```

3.2 Proof of correctness

We provide the proof of correctness and time complexity for the proposed modification in this section.

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

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 $P[l,l'] = \{(B,C)|a_{l+1}...a_{l'} \in L(B)L(C)\}$. Further, algorithm computes $f(P[l,l']) = \{A|a_{l+1}...a_{l'} \in L(A)\}$, 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 $\{(B,C)|\exists(\frac{l+m}{2} \leq k < l') : a_{i+1}...a_k \in L(B), a_{k+1}...a_j \in L(C)\}$ and $\forall (i,j) \in right = \{(\frac{l+m}{2}, m, \frac{l'+m'}{2}, m')\}$ all pairs $\{(B,C)|\exists(m \leq k < \frac{l'+m'}{2}) : a_{i+1}...a_k \in L(B), a_{k+1}...a_j \in L(C)\}$. 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 $\{(B,C)|\exists(\frac{l+m}{2} \leq k < m)$ and $(l' \leq k < \frac{l'+m'}{2}) : a_{i+1}...a_k \in L(B), a_{k+1}...a_j \in L(C)\}$. 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,* $T[i,j] = \{A | a_{i+1}...a_j \in L(A)\}$ $\forall l \leq i < j < m, l' \leq i < j < m'$ *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* $P[i,j] = \{(B,C) | \exists (m \leq k < l') : a_{i+1}...a_k \in L(B), a_{k+1}...a_j \in L(C)\}$ $\forall l \leq i < m, l' \leq j < m'$ $\forall (l,m,l',m') \in M$.

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 *costructLayer(i)* returns $2^{k-i} - 1$ matrices of size 2^i.

Lemma 1.

- $\forall i \in \{1,..,k-1\}$ $\sum |layer|$ *for the calls of completeVLayer(layer) where* $\forall (l,m,l',m') \in layer$ *with* $m - l = 2^{k-i}$ *is exactly* $2^{2i-1} - 2^{i-1}$*;*
- $\forall i \in \{1,..,k-1\}$ *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 $\forall i \in \{1,..,j\}$ $\sum |layer|$ 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 [12], 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 problem hat arose at the intersection of the theory of formal languages and bioinformatics – 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 $\leq 2^{l'}$, where $2^{l'-2} < s \leq 2^{l'-1}$.

$l' = k - (m-2)$, $(m-2) = k - l'$

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

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 $\frac{n}{2}$, 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 << n than the original algorithm.

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 [13]. 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. Chomsky, N.: On certain formal properties of grammars. Information and control **2**(2), 137–167 (1959)
2. Chomsky, N. and Schtzenberger 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)
3. Rivas, E. and Eddy, S. R: The language of RNA: a formal grammar that includes pseudoknots. Bioinformatics **16**(4), 334-340 (2000)
4. Knudsen, B. and Hein, J.: RNA secondary structure prediction using stochastic context-free grammars and evolutionary history. Bioinformatics (Oxford, England) **15**(6), 446-454 (1999)
5. Yuan, C. and Lei, J. and Cole, J. and Sun, Y.: Reconstructing 16S rRNA genes in metagenomic data. Bioinformatics **31**(12), i35-i43 (2015)
6. Dowell, R. D. and Eddy, S. R.: Evaluation of several lightweight stochastic context-free grammars for RNA secondary structure prediction. BMC bioinformatics **5**(1), 71 (2004)
7. 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)
8. Kasami, T.: An efficient recognition and syntax-analysis algorithm for context-free languages. Coordinated Science Laboratory Report no. R-257 (1966)
9. Younger, D.H.: Recognition and parsing of context-free languages in time n3. Information and control **10**(2), 189–208 (1967)
10. Earley, J.: An efficient context-free parsing algorithm. Communications of the ACM **13**(2), 94–102 (1970)
11. Valiant, L. G.: General context-free recognition in less than cubic time. Journal of computer and system sciences **10**(2), 308–315 (1975)
12. Okhotin, A.: Parsing by matrix multiplication generalized to Boolean grammars. Theoretical Computer Science **516**, 101–120 (2014)
13. The M4RI Library – Version 20121224, http://m4ri.sagemath.org. Last accessed 5 Jan 2019