Presentation on DNA sequence Compression using the Burrows-Wheeler Transform

A Demonstration by

Shrestha Thumati
EE18BTECH11041
Adyasa Mohanty
EE18BTECH11048

March 28, 2021

The Problem Statement

DNA (Deoxyribonucleic acid) is a molecule that contains the biological instructions that make each species unique.
It is estimated that the number of available nucleotide bases doubles approximately every 14 months, while genomes are being sequenced at a rate of 15 complete genomes per month.
Hence there is a need of developing effective techniques for the management, organization, and distribution of this unprecedented mass of biological sequence data.

Introduction

- DNA contains four basic nucleaotides **adenine**, **cytosine**, **guanine**, and **thymine**, usually abbreviated using the symbols **A**, **C**, **G** and **T** respectively.
- A biological sequence is a one-dimensional sequence of symbols, for instance with an alphabet of 4 symbols for DNA, or RNA, and 20 symbols for proteins.
- Repetition and regularities are very prominent in biological sequences.
- Tandem repeats and interspersed repeats are prominent in DNA.

General Data Compression

- There are three types of lossless compression schemes: **symbol-wise substitution**(Huffman), **dictionary- based**(LZ78,LZ77) and **context-based methods**(PPM).

- Block-sorting methods also make use of contexts, but here the contexts are characters permuted in such a way that the symbols in lexicographically similar contexts will be clustered together. The block sorting approach is also called **Burrows-Wheeler Transform** or **BWT** for short.

Compression of Biological Sequences

- Traditional algorithms like Arithmetic, Huffman and LZ based method for biological sequence compression result in data expansion instead of compression.
- These algorithms do not consider the regularities present in biological sequences.
- Compresssion algorithms for biological sequences like BIOCOMPRESS 1, BIOCOMPRESS 2 consider repetioin structures like **tandem repeats**, **palindromes**, **complemented repeats**, and **complemented palindromes**.

The Solution

- This where the Burrow-Wheeler Transformation steps in. It is an algorithm that helps in compression when the data has huge amount of repetition.
  Next we explain two kinds of parsing techniques that are suitable for this problem.

Burrows-Wheeler Transform

- The Burrows–Wheeler transform is an algorithm used to prepare data for use with data compression techniques.
- Its computation complexity is $O(n)$ and its space complexity is also $O(n)$.
- It performs cyclic rotation of the characters in the sequence, such that characters in lexically similar contexts will be near to each other.

Burrows-Wheeler Transform

- The success of this transform depends upon one value having a high probability of occurring before a sequence, so that in general it needs fairly long samples (a few kilobytes at least) of appropriate data (such as text).
- BWT is completely reversible, allowing the original document to be re-generated from the last column data.

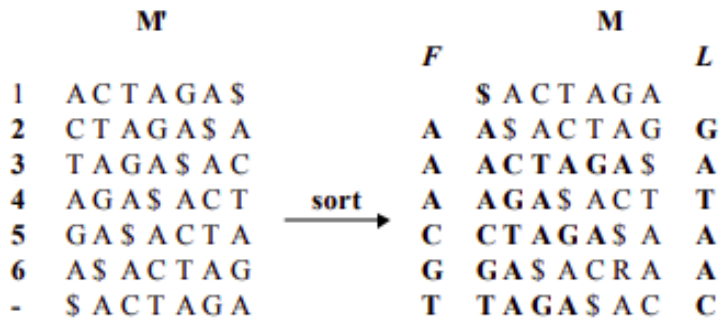
The Forward BWT

Given an input sequence $T = t_1, t_2, ..., t_u$

- **Step-1**
  Form u permutations of T by cyclic rotations of the characters in T. The permutations form a uu matrix $M^{'}$.
- **Step-2** Sort the rows of M' lexicographically to form another matrix M.
- **Step-3** Record L, the last column of the sorted permutation matrix M, and id, the row number for the row in M that corresponds to the original sequence T.
- The output of the BWT is the pair, **(L, id)**.

Forward BWT

For example, suppose $T = ACTAGA$.



The output of the transformation will be the pair: **(L, id)=(GATAAC, 2)**

The Inverse BWT

- **Step-1**
  Sort L to produce F, the array of first characters.
- **Step-2**
  Compute V, the transformation vector that provides a one-to-one mapping between the elements of L and F, such that $F[V[j]] = L[j]$.
- **Step-3**
  Generate the original sequence T, since the rows in M are cyclic rotations of each other, the symbol $L[i]$ cyclically precedes the symbol $F[i]$ in T. That is, $L[V[j]]$ cyclically precedes $L[j]$ in T.
  For the example with $T = ACTAGA$, we will have $\mathbf{V = [5\ 1\ 6\ 2\ 3\ 4]}$. Given V and L, we can generate the original text by iterating with V.

BWT Compression Pipeline

input $\rightarrow$**BWT** $\rightarrow$**MTF** $\rightarrow$**RLE** $\rightarrow$**VLC** $\rightarrow$**output**

MTF : Data transformation algorithm that restructures data in such a way that the transformed message is more compressible and therefore used as an extra step in compression.
RLE : Lossless Data Compression Algorithm
VLC : Variable Length Coding

Parsing Algorithm VPS1

- **VPS1** provides offline dictionary compression with pointers in the dictionary
- **Repetition codes**: **1**: repeat; **2** reverse repeat; **3**: palindrome; **4**: compliment; **5**: complimented palindrome; **6**: reverse compliments

VPS1

General structure of dictionary :

index	r	$l(r)$	$rT(r)$	$\eta(r)$	**positions**
i	r_i	$l(r_i)$	1	$\eta(r_i,1)$	p(i,1,1), p(i,1,2) …
			2	$\eta(r_i,2)$	p(i,2,1), p(i,2,2) …
			3	$\eta(r_i,3)$	p(i,3,1), p(i,3,2) …
			4	$\eta(r_i,4)$	p(i,4,1), p(i,4,2) …
			5	$\eta(r_i,5)$	p(i,5,1), p(i,5,2) …

r_i i-th repetition pattern; $\mathbf{l(r)=\|r\|}$ length of repeat pattern; $\mathbf{rT(r)}$ repetition code; $\boldsymbol{\eta}\mathbf{(r)}$ is the total number of occurrence of r; $\boldsymbol{\eta}\mathbf{(r, j)}$ is the number of occurrences of r with repetition type j; **p(i,j,k)** is the position in the sequence of the k-th instance of repetition pattern r_i , with repetition type j.

VPS1

Example

	P1		P2		P3		P4		P5		P6		P7
x_1	AACTGTCAA	x_2	AA	x_3	GTCAA	x_4	TG	x_5	AACTG	x_6	TTGACAGT	x_7	AA

index	r	l(r)	t(r)	$\eta(r)$	positions
1	AACTGTCAA	9	1	1	P_1
			5	1	P_6
2	GTCAA	5	1	1	P_3
			2	1	P_5
3	AA	2	1	2	P_2, P_7
4	TG	2	1	2	P_4

Parsed sequence :

$$\textbf{Parse(S)} : x_1x_2x_3x_4x_5x_6x_7$$

Encoding of Vocabulary

The term vocabulary is used to trefer to the esemble of repeat structures.

Vocabulary Encoding A:

Vocabulary: $r_1\ \mathbf{0}\ r_2\ \mathbf{0} \ldots \mathbf{0}\ r_\kappa\ \mathbf{0}$

Positions:

$$P_{1,1}, P_{1,2}, \ldots P_{1,\eta(1)}\ \mathbf{0}\ P_{2,1}, P_{2,2}, \ldots P_{2\eta(2)}\ \mathbf{0} \ldots \mathbf{0} P_{\kappa,1}, P_{\kappa,2}, \ldots P_{\kappa,\eta(\kappa)}\ \mathbf{0}$$

$P_{r,j}$ =position of the j-th occurrence of repeat pattern r; K = dictionary size; u = $\|S\|$ = size of the sequence; $\|\sum\| = 4$ for nucleotides;$\beta = log\|\sum\|$; b(n) = log(n)

Cost of Vocabulary

Cost of **original sequence**:

$$C(S) = |S|.\lceil \log \Sigma \rceil = u.\beta$$

Cost of **parsed sequence**

$$C(parse(S)) = u\beta - \beta \sum_{i=1}^{\kappa} l(r_i)\eta(r_i)$$

Cost of **vocabulary**:

$$C(V_A(S)) = \beta_1 \sum_{i=1}^{\kappa} (l(r_i)+1) = \kappa\beta_1 + \beta_1 \sum_{i=1}^{\kappa} l(r_i),$$

$\beta < \beta_1 < \beta$

Cost of Vocabulary

Cost of **positions**:

$$C(Posn(S)) = \sum_{i=1}^{\kappa} \sum_{j=1}^{\eta(i)} b(P_{i,j}) + \sum_{i=1}^{\kappa} 1 = \kappa + \sum_{i=1}^{\kappa} \sum_{j=1}^{\eta(i)} \lceil \log P_{i,j} \rceil$$

The **cost of dictionary** representation is then the combined cost of the vocabulary and the positions:

$$C(D(S)) = C(V_A(S)) + C(Posn(S))$$

Compression Gain:

$$G(\hat{S}) = C(S) - \left[C(D(S)) + C(Parse(S)) \right]$$

Parsing Algorithm VPS2

Identify the different repetition structures, and replace their positions in the sequence with reference indices indicating their positions in the dictionary, and the type of repeat.

VPS2 Parsing Schemes

Parsing schema1: $\langle referenceindex \rangle < rI >$
Parsing schema2: $\langle referenceindex, repeattype \rangle < rI, rT >$
Parsing schema3: $\langle index, repeattype, range \rangle < rI, rT, sP, nP >$
In **fixed-length** parsing, each reference code in the sequence must have the same number of parameters.
In **variable-length** parsing, reference codes in the sequence are allowed to have different number of parameters.

VPS2

Example

	P1		P2		P3		P4		P5		P6		P7
x_1	AACTGTCAA	x_2	AA	x_3	GTCAA	x_4	TG	x_5	AACTG	x_6	TTGACAGT	x_7	AA

rI	repeat, r
1	AACTGTCAA
2	GTCAA
3	AA
4	TG

Fixed Length Parsing

Parse(S): x_1<1,1>x_2<3,1>x_3<2,1>x_4<4,1>x_5<2,2>x_6<1,5>x_7<3,1>

Variable Length Parsing

Parse(S): x_1<1>x_2<3>x_3<2>x_4<4>x_5<1,1,1,5>x_6<1,5>x_7<3>

VPS1 vs VPS2

l(r)is the length of repeat pattern
h(r) is the total number of occurrence of r

Algorithm VPS1 : simple remaining parsed sequence, but a complicated dictionary. **l(r)** and **h(r)** need to be included in the dictionary.

Algorithm VPS2 : the simplicity of the dictionary, **l(r)** and **h(r)** need not be included in the dictionary.

Conclusion

- Biological sequence data like DNA cannot be performed by just using any one algorithm, we need to combine a number of algorithms to perform an optimal compression
- Compared to dictionary based methods like LZ, context based methods like PPM,BWT are preferred.
- The BWT is second to the PPM in compaction performance, but faster than the PPM schemes.

The End