BWT - Burrows Wheeler Transform

A block-sorting lossless data compression algorithm

- Introduction
- Ciphering phase
- Deciphering phase
- Deciphering phase – improvements
- Relevant compression techniques
- Effectiveness of BWT
- Comparison with other compression programmes
- References

Alberto Chiusole, s249223
Advanced Programming and Algorithmic Design @ uniTS – A.Y. 2017-2018

Introduction to BWT

- Used to preprocess strings before actual compression
- Produces output with high Index of Coincidence (IC)

==

Identical characters are often in groups

- Compression is easy, i.e. with *run-length* encoding

Ciphering phase (1)

- Encode the string `mississippi`, length `N`
- Add separator symbol (End Of Line) at the end; it must not be present in the string
- `S = 'mississippi' + $`

Ciphering phase (2)

- Create all the cyclic rotations/circular shifts of the string `S`
- Single character permutations; move the first character after the last

```
m i s s i s s i p p i $
i s s i s s i p p i $ m
s s i s s i p p i $ m i
s i s s i p p i $ m i s
i s s i p p i $ m i s s
s s i p p i $ m i s s i
s i p p i $ m i s s i s
i p p i $ m i s s i s s
p p i $ m i s s i s s i
p i $ m i s s i s s i p
i $ m i s s i s s i p p
$ m i s s i s s i p p i
```

Ciphering phase (3)

- Sort all the permutations from top to bottom
- F and L are the First and Last columns of the matrix

```
                                  F                     L
m i s s i s s i p p i $           $ m i s s i s s i p p i
i s s i s s i p p i $ m           i $ m i s s i s s i p p
s s i s s i p p i $ m i           i p p i $ m i s s i s s
s i s s i p p i $ m i s           i s s i p p i $ m i s s
i s s i p p i $ m i s s           i s s i s s i p p i $ m
s s i p p i $ m i s s i   --->    m i s s i s s i p p i $
s i p p i $ m i s s i s           p i $ m i s s i s s i p
i p p i $ m i s s i s s           p p i $ m i s s i s s i
p p i $ m i s s i s s i           s i p p i $ m i s s i s
p i $ m i s s i s s i p           s i s s i p p i $ m i s
i $ m i s s i s s i p p           s s i p p i $ m i s s i
$ m i s s i s s i p p i           s s i s s i p p i $ m i
```

Ciphering phase (4)

- Take only the last column `` `L = ipssm$pissii` ``
- Encoding done!

```
m i s s i s s i p p i $
i s s i s s i p p i $ m
s s i s s i p p i $ m i
s i s s i p p i $ m i s
i s s i p p i $ m i s s
s s i p p i $ m i s s i
s i p p i $ m i s s i s
i p p i $ m i s s i s s
p p i $ m i s s i s s i
p i $ m i s s i s s i p
i $ m i s s i s s i p p
$ m i s s i s s i p p i
```

```
F                     L
$ m i s s i s s i p p i
i $ m i s s i s s i p p
i p p i $ m i s s i s s
i s s i p p i $ m i s s
i s s i s s i p p i $ m
m i s s i s s i p p i $
p i $ m i s s i s s i p
p p i $ m i s s i s s i
s i p p i $ m i s s i s
s i s s i p p i $ m i s
s s i p p i $ m i s s i
s s i s s i p p i $ m i
```

Deciphering phase (1)

- The decipher only sees the encoded string `L` but,
- sort `L`, obtain `F = sort(L) = $iiiimppssss`

```
F                     L
$ m i s s i s s i p p i
i $ m i s s i s s i p p
i p p i $ m i s s i s s
i s s i p p i $ m i s s
i s s i s s i p p i $ m
m i s s i s s i p p i $
p i $ m i s s i s s i p
p p i $ m i s s i s s i
s i p p i $ m i s s i s
s i s s i p p i $ m i s
s s i p p i $ m i s s i
s s i s s i p p i $ m i
```

Deciphering phase (2)

- To rebuild the initial matrix:

  stick together the columns `L-F`

```
F                     L
$ m i s s i s s i p p i
i $ m i s s i s s i p p
i p p i $ m i s s i s s
i s s i p p i $ m i s s
i s s i s s i p p i $ m
m i s s i s s i p p i $
p i $ m i s s i s s i p
p p i $ m i s s i s s i
s i p p i $ m i s s i s
s i s s i p p i $ m i s
s s i p p i $ m i s s i
s s i s s i p p i $ m i
```

```
L F
i $
p i
s i
s i
m i
$ m
p p
i p
s s
s s
i s
i s
```

Deciphering phase (3)

- The 2-mer `L-F` is made of substrings of `S`
- Sort lexicographically → first 2 columns original matrix!

```
F                       L        L F               F
$ m i s s i s s i p p   i        i $               $ m
i $ m i s s i s s i p   p        p i               i $
i p p i $ m i s s i s   s        s i               i p
i s s i p p i $ m i s   s        s i               i s
i s s i s s i p p i $   m        m i     sort      i s
m i s s i s s i p p i   $  --->  $ m     --->      m i
p i $ m i s s i s s i   p        p p               p i
p p i $ m i s s i s s   i        i p               p p
s i p p i $ m i s s i   s        s s               s i
s i s s i p p i $ m i   s        s s               s i
s s i p p i $ m i s s   i        i s               s s
s s i s s i p p i $ m   i        i s               s s
```

Deciphering phase (4)

- Take the 3-mer made of the first 2 columns + `L`
- Sort lexicographically → first 3 columns original matrix!

```
F                     L
$ m i s s i s s i p p i
i $ m i s s i s s i p p
i p p i $ m i s s i s s
i s s i p p i $ m i s s
i s s i s s i p p i $ m
m i s s i s s i p p i $
p i $ m i s s i s s i p
p p i $ m i s s i s s i
s i p p i $ m i s s i s
s i s s i p p i $ m i s
s s i p p i $ m i s s i
s s i s s i p p i $ m i
```

```
L F
i $ m
p i $
s i p
s i s
m i s
$ m i
p p i
i p p
s s i
s s i
i s s
i s s
```

sort

```
F
$ m i
i $ m
i p p
i s s
i s s
m i s
p i $
p p i
s i p
s i s
s s i
s s i
```

Deciphering phase (5)

- Repeat until the original matrix is complete
- The string `S` is the one ending with `$`

- Simple approach, but poor performances:
  - `O(N²)` *space* consumption for storing the `N²` matrix
  - `O(N² logN)` computations: `N` *k*-mers sorting operations with a `O(logN)` sorting algorithm

Deciphering phase - improvements (1)

- Curious *first-last* property:
- The *relative* order in the group of same letters is preserved: observe, e.g., the relative order of the `` `i` ``’s

F		L
\$	m i s s i s s i p p	$\mathbf{i}_1$
$\mathbf{i}_1$	\$ m i s s i s s i p	p_1
$\mathbf{i}_2$	p p i \$ m i s s i s	s_1
$\mathbf{i}_3$	s s i p p i \$ m i s	s_2
$\mathbf{i}_4$	s s i s s i p p i \$	m_1
m_1	i s s i s s i p p i	\$
p_1	i \$ m i s s i s s i	p_2
p_2	p i \$ m i s s i s s	$\mathbf{i}_2$
s_1	i p p i \$ m i s s i	s_3
s_2	i s s i p p i \$ m i	s_4
s_3	s i p p i \$ m i s s	$\mathbf{i}_3$
s_4	s i s s i p p i \$ m	$\mathbf{i}_4$

Proof:

- consider rows starting with a specific letter
- delete the first letter, the columns are still sorted
- apply those first letter after `` `L` ``, columns are still sorted
- these new rows are available inside the matrix, and they are still sorted relatively to one another

Deciphering phase - improvements (2)

- Start from the EOL symbol `$` in `F`, go backwards into `L`: find `i`$_1$
- Search for `i`$_1$ in `F`, go backwards in `L` and find `p`$_1$

F		L
$	m i s s i s s i p p	**i_1**
i_1	$ m i s s i s s i p	**p_1**
i_2	p p i $ m i s s i s	s_1
i_3	s s i p p i $ m i s	s_2
i_4	s s i s s i p p i $	m_1
m_1	i s s i s s i p p i	$
p_1	i $ m i s s i s s i	p_2
p_2	p i $ m i s s i s s	i_2
s_1	i p p i $ m i s s i	s_3
s_2	i s s i p p i $ m i	s_4
s_3	s i p p i $ m i s s	i_3
s_4	s i s s i p p i $ m	i_4

S = ...**pi**

Deciphering phase - improvements (3)

- Search for \`$p_1$\` in \`F\`, go backwards in \`L\` and find \`$p_2$\`
- Search for \`$p_2$\` in \`F\`, go backwards in \`L\` and find \`$i_2$\`

F		L
$	m i s s i s s i p p	i_1
i_1	$ m i s s i s s i p	p_1
i_2	p p i $ m i s s i s	s_1
i_3	s s i p p i $ m i s	s_2
i_4	s s i s s i p p i $	m_1
m_1	i s s i s s i p p i	$
$\mathbf{p_1}$	i $ m i s s i s s i	$\mathbf{p_2}$
$\mathbf{p_2}$	p i $ m i s s i s s	$\mathbf{i_2}$
s_1	i p p i $ m i s s i	s_3
s_2	i s s i p p i $ m i	s_4
s_3	s i p p i $ m i s s	i_3
s_4	s i s s i p p i $ m	i_4

S = ...**ippi**

Deciphering phase - improvements (4)

- No need to rebuild the original matrix
- Linear memory consumption: `2 N → O(N)`
- Linear number of computations to build the 2 first-last arrays

Relevant compression techniques (1)

- *Move-to-front transform* encoding can be applied after BWT and can improve Index Coincidence
- using the alphabet `a..z`, take a string (e.g., `S = `ipssm$pissii``), note the index of each element

  but also modify the alphabet by moving to the front the last char of the alphabet used
- There will be lots of low index values

MTF(S, “a..z+$”) = [8, 15, 18, 0, 14, 26, 3, 4, 4, 0, 1, 0]

Relevant compression techniques (2)

- An encoded `L` string can be compressed with, e.g., *run-length encoding*.
- For long strings, it reduces the memory needed, especially with short alphabets (e.g., DNA strings)

```
RLE('ipssm$pissii') = i1 p1 s2 m1 $1 p1 s2 i2
```

Effectiveness of BWT

- Example: Normal text contains many occurrences of the word `the`
- create the BWT matrix -> lots of rows starting with `he` and ending with `t`

mlpeeHehhemmigsdmgrgfmeeeeeeetmierpvttpsssHHchcmthllltlrrcvivfnteleDeMMMMMMMMwlphmwhhmMhpw
chtwmmempwWrWwrrWrtlcehhsthhhhhhhhhhhhhhbwwwwtlttrbfhhHfrrrrrrsmeHeerllwdmldwtwwdrmmibrmumiua
nnnnonirnrnaieutiuiiuaiiiutttrsiiaieseieaienaa'eeeoleen'nennnlnninna'n'annineno'a'nnannoiiinonnnnnne''n'nnnnnn
o'oreneanseiiidnnoelnnnrsvthhhhchhrltrlnhuhcnhmhhhlmlggchmskhhhhmvslhsgshrtkhhhhhhhhhrhbctspmdvgce
hhnkthhhhhhhHsheebrhmmhhrhrhkkkkdrlktggukbhshchhmc nmtdlrfrrercsyltgkmPdhddurpppppsfpp lwhhr
rjrtidigtvmtdlchhhSwtbhriBbrfsesbwWtWfferld vvvzmh'vehmtufDst tmvrNhhpbhvhrhhwdBBBBBBhihh hhhtuttg
svtmmmoodvtyrhdrrlli uujlilepggmmremrii d ooooooooeoOooooooooOOoofol i r fooin ee n nnnnnnnnnn
aAnaaarrnanooiuiiiiuiiiiinoiicccctgcttctctctccttcgttcttcttcccctccSWTttTtWwWtttTtttttttttttTtttttttttttttttttttTtttttttttsWw
tttttswtTntwwWwwwctttsttTtttttttttwWwcttWwwGTtttttgggggggTwWWtrtlgswhhhhhhsrtrtDaDDDsvrrfhftdollthnm
nmmnresffllllllwtoahlttnaattttmmKhmdKkkKKbphhaaottttttnttlhaspfkhhhhhHThhhhhhhhTh h hDruhlox s
erwwwWwwwWblltdebarsrcalnroaaonaaacraeiaiiiiiaoisloclllllalllallouousurabbcttbraspwtmmeaerccuaaeeeieiaee
uaaobiufoitrroeioioonoiooaeaoaioaoroyeooioooaaelsruooeiiioroelooeaieaeoeewaeaogioeaiioieIIiioeIiIowirwgioo
oweeueiiiieaoaAaaaaAAaaaaaaaaaAaaaAaaaaaaaAaaiaaueoaaaooiuiiiuoiiiiiiiaaiaiuiiieankkkkoaioeioaeaaiaeaU
ueueetStShtsSnnttttttttTtstttthsiiitsiooooGbDgtTlmtjmoopoPhrsrtrRcccssRcCssccccconiiipiwggrcimfGttllLLdcFfF
HHHHHHHHrHHHHfsmffffwwffspFFFFNNNthhhhhrmmMmmphlhnnnnnnnnmmDd mhhyyhYrvrhwyc m
hsbithrcnnHnnNnt nrdrnumrme osssssppsmommsU aa m r miuemisn naiaeueoeuoeuuueeuouuaauuuueeaa

BWT of ~700 words of Shakespeare’s Hamlet

Comparison with other compression progr.

Program name	CPU Time (s) Compression	Decompression	Average bit/char
compress	9.6	5.2	3.63
gzip	42.6	4.9	2.71
BWT + Huffman coder	*51.1*	*9.4*	*2.43*
comp-2	603.2	614.1	2.47

BWT is widely used, e.g., in the bzip2 compressor, for its efficiency and great advantages.

References

- Burrows, Michael; Wheeler, David J. (1994), A block sorting lossless data compression algorithm, Technical Report 124, Digital Equipment Corporation