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

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 p

$ m i s s i s s i p p

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

Ciphering phase (3)

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

```
mississippi$
                   $mississippi
ississippi$m
                   i$mississipp
                   ippi$mississ
ssissippi$mi
sissippi$mis
                   issippi$miss
                   ississippi$m
issippi$miss
                   mississippi$
ssippi$missi
sippi$missis
                   pi$mississip
ippi$mississ
                   ppi$mississi
ppi$mississi
                   sippi$missis
                   sissippi$mis
pi$mississip
                   ssippi$missi
i$mississipp
$mississippi
                   ssissippi$mi
```

Ciphering phase (4)

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

```
mississippi$
                   $mississippi
ississippi$m
                   i$mississipp
                 ippi$mississ
ssissippi$mi
sissippi$mis
                   issippi$miss
                   ississippi$m
issippi$miss
                   mississippi$
ssippi$missi
                   pi$mississip
sippi$missis
ippi$mississ
                   ppi$mississi
ppi$mississi
                   sippi$missis
pi$mississip
                   sissippi$mis
                   ssippi$missi
i$mississipp
$mississippi
                   ssissippi$mi
```

Deciphering phase (1)

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

```
$ mississippi
i$mississipp
ippi$mississ
issippi$miss
ississippi$m
mississippi$
pi$mississip
ppi$mississi
sippi$missis
sissippi$mis
s s i p p i $ m i s s i
s s i s s i p p i $ m i
```

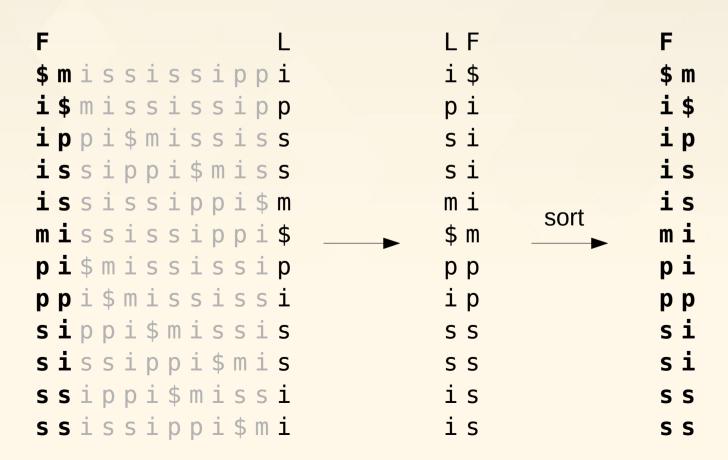
Deciphering phase (2)

To rebuild the initial matrix:
 stick together the columns `L-F`

```
LF
                      i$
$mississippi
i $ m i s s i s s i p p
                      рi
                      si
ippi$mississ
issippi$miss
                      si
ississippi$m
                      тi
mississippi$
                      $ m
pi$mississip
                      p p
                      iр
ppi$mississi
sippi$missis
                      SS
sissippi$mis
                      SS
ssippi$missi
                      is
ssissippi$mi
                      is
```

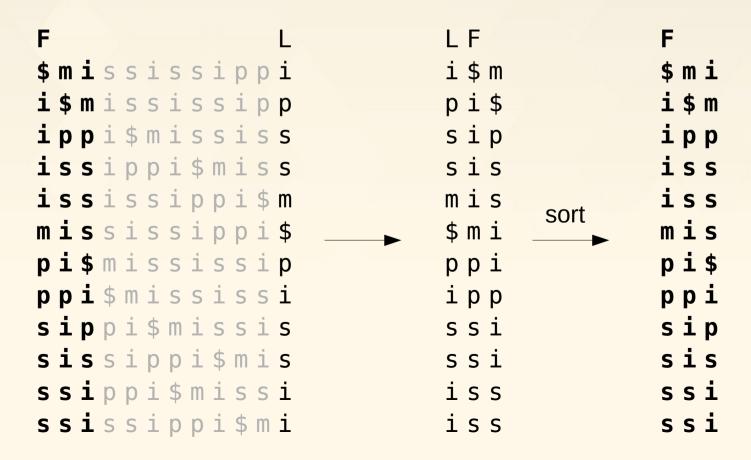
Deciphering phase (3)

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



Deciphering phase (4)

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



Deciphering phase (5)

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

- Simple approach, but poor performances:
 - 0 (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

```
$ mississippi,
i, $mississipp
i, ppi$mississ,
i, ssippi$miss,
i<sub>4</sub> ssissippi$m<sub>1</sub>
m_1 ississippi$
p_1 i $ m i s s i s s i p_2
p, p i $ m i s s i s s i,
S_1 i p p i $ m i s s i S_3
s_2 is sippi$mis<sub>4</sub>
s, sippi$missi,
S_a s i s s i p p i $ m \mathbf{i}_a
```

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 `i₁`
- Search for `i₁` in `F`, go backwards in `L` and find `p₁`

```
$ mississippi,
i, $ m i s s i s s i p p,
i, ppi$mississ,
i, s s i p p i $ m i s s,
i_{a} s s i s s i p p i \$ m<sub>1</sub>
m_1 ississippi$
p_1 i $ m i s s i s s i p_2
p<sub>2</sub>pi$mississi<sub>2</sub>
S_1 i p p i $ m i s s i S_3
s_2 is sippi$mis<sub>4</sub>
S<sub>3</sub> sippi$missi<sub>3</sub>
S_a s i s s i p p i $ m i_a
```

$$S = \dots pi$$

Deciphering phase - improvements (3)

- Search for `p₁` in `F`, go backwards in `L` and find `p₂`
- Search for `p₂` in `F`, go backwards in `L` and find `i₂`

```
$ mississippi,
i, $ m i s s i s s i p p,
i, ppi$mississ,
i, s s i p p i $ m i s s,
i_{a} s s i s s i p p i \$ m<sub>1</sub>
m_1 ississippi$
p_1 i $ m i s s i s s i p_2
p<sub>2</sub>pi$mississi,
S_1 i p p i $ m i s s i S_3
s_2 is sippi$mis<sub>4</sub>
S<sub>3</sub> sippi$missi<sub>3</sub>
S_a s i s s i p p i $ m i_a
```

$$S = \dots ippi$$

Deciphering phase - improvements (4)

- No need to rebuild the original matrix
- Linear memory consumption: 2 N → 0(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`

Comparison with other compression progr.

Program name	CPU T Compression	ime (s) 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