The Burrows-Wheeler Transform between Data Compression and Combinatorics on Words

Giovanna Rosone and Marinella Sciortino

Dipartimento di Matematica e Informatica
University of Palermo, ITALY

CiE 2013

1. Introduction
   - Preliminaries

2. The Burrows Wheeler Transform
   - How does BWT work?
   - How computing the BWT?
   - Applications
   - Combinatorial Issues on the BWT
   - Effects of the combinatorial properties

3. The Extended Burrows Wheeler Transform
   - How does EBWT work?
   - How computing the EBWT?
   - Applications

4. Further works

Preliminaries

- Let Σ denote a non-empty finite alphabet.
- A word w over an alphabet Σ is a finite sequence of letters of Σ. We denote by Σ^* the set of all words over Σ.
- Given a finite word $w = a_1 a_2 \cdots a_n$, $a_i \in \Sigma$, a factor of w is written as $w[i,j] = a_i \cdots a_j$. A factor $w[1,j]$ is called a prefix, while a factor $w[i,n]$ is called a suffix.
- A non-empty word $w \in \Sigma^*$ is *primitive* if $w = u^h$ implies $w = u$ and $h = 1$.
- Two words $u, v \in \Sigma^*$ are conjugate, if $u = xy$ and $v = yx$ for some $x, y \in \Sigma^*$. Thus conjugate words are just cyclic shifts of one another.
- A *Lyndon word* is a primitive word which is the minimum among its conjugates, with respect to the lexicographic order relation.
  - $mathematics$ is not a Lyndon word
  - $athematicsm$ is a Lyndon word.

Preliminaries

- Let Σ denote a non-empty finite alphabet.
- A word w over an alphabet Σ is a finite sequence of letters of Σ. We denote by Σ^* the set of all words over Σ.
- Given a finite word $w = a_1 a_2 \cdots a_n$, $a_i \in \Sigma$, a factor of w is written as $w[i,j] = a_i \cdots a_j$. A factor $w[1,j]$ is called a prefix, while a factor $w[i,n]$ is called a suffix.
- A non-empty word $w \in \Sigma^*$ is *primitive* if $w = u^h$ implies $w = u$ and $h = 1$.
- Two words $u, v \in \Sigma^*$ are conjugate, if $u = xy$ and $v = yx$ for some $x, y \in \Sigma^*$. Thus conjugate words are just cyclic shifts of one another.
- A *Lyndon word* is a primitive word which is the minimum among its conjugates, with respect to the lexicographic order relation.
  - $mathematics$ is not a Lyndon word
  - $athematicsm$ is a Lyndon word.

Preliminaries

- Let Σ denote a non-empty finite alphabet.
- A word w over an alphabet Σ is a finite sequence of letters of Σ. We denote by Σ^* the set of all words over Σ.
- Given a finite word $w = a_1 a_2 \cdots a_n$, $a_i \in \Sigma$, a factor of w is written as $w[i,j] = a_i \cdots a_j$. A factor $w[1,j]$ is called a prefix, while a factor $w[i,n]$ is called a suffix.
- A non-empty word $w \in \Sigma^*$ is *primitive* if $w = u^h$ implies $w = u$ and $h = 1$.
- Two words $u, v \in \Sigma^*$ are conjugate, if $u = xy$ and $v = yx$ for some $x, y \in \Sigma^*$. Thus conjugate words are just cyclic shifts of one another.
- A *Lyndon word* is a primitive word which is the minimum among its conjugates, with respect to the lexicographic order relation.
  - $mathematics$ is not a Lyndon word
  - $athematicsm$ is a Lyndon word.

The Burrows Wheeler Transform: the goal

The Burrows Wheeler Transform (BWT) is a reversible transformation that produces a permutation $bwt(w)$ of an input sequence w, defined over an ordered alphabet Σ, so that occurrences of a given symbol tend to occur in clusters in the output sequence.

The Burrows Wheeler Transform

Given a word $w \in \Sigma^*$, $bwt(w)$ is a permutation of w, obtained as concatenation of the last letters of the lexicographically sorted list of its conjugates.

Example: $w = mathematics$

$m\ a\ t\ h\ e\ m\ a\ t\ i\ c\ s$

1	a	t	h	e	m	a	t	i	c	s	m
2	a	t	i	c	s	m	a	t	h	e	m
3	c	s	m	a	t	h	e	m	a	t	i
4	e	m	a	t	i	c	s	m	a	t	h
5	h	e	m	a	t	i	c	s	m	a	t
6	i	c	s	m	a	t	h	e	m	a	t
7	m	a	t	h	e	m	a	t	i	c	s
8	m	a	t	i	c	s	m	a	t	h	e
9	s	m	a	t	h	e	m	a	t	i	c
10	t	h	e	m	a	t	i	c	s	m	a
11	t	i	c	s	m	a	t	h	e	m	a

Each row of M is a conjugate of w in lexicographic order.
The index I is the row of M containing the original word.
$bwt(w) = L = mmihttsecaa$ and $I = 7$.

The Burrows Wheeler Transform

Given a word $w \in \Sigma^*$, $bwt(w)$ is a permutation of w, obtained as concatenation of the last letters of the lexicographically sorted list of its conjugates.

Example: $w = mathematics$

$m\ a\ t\ h\ e\ m\ a\ t\ i\ c\ s$

1	a	t	h	e	m	a	t	i	c	s	m
2	a	t	i	c	s	m	a	t	h	e	m
3	c	s	m	a	t	h	e	m	a	t	i
4	e	m	a	t	i	c	s	m	a	t	h
5	h	e	m	a	t	i	c	s	m	a	t
6	i	c	s	m	a	t	h	e	m	a	t
7	m	a	t	h	e	m	a	t	i	c	s
8	m	a	t	i	c	s	m	a	t	h	e
9	s	m	a	t	h	e	m	a	t	i	c
10	t	h	e	m	a	t	i	c	s	m	a
11	t	i	c	s	m	a	t	h	e	m	a

Each row of M is a conjugate of w in lexicographic order.
The index I is the row of M containing the original word.
$bwt(w) = L = mmihttsecaa$ and $I = 7$.

The Burrows Wheeler Transform

Given a word $w \in \Sigma^*$, $bwt(w)$ is a permutation of w, obtained as concatenation of the last letters of the lexicographically sorted list of its conjugates.

Example: $w = mathematics$

$$
\begin{array}{ccccccccccc}
m & a & t & h & e & m & a & t & i & c & s \\
a & t & h & e & m & a & t & i & c & s & m \\
t & h & e & m & a & t & i & c & s & m & a \\
h & e & m & a & t & i & c & s & m & a & t \\
e & m & a & t & i & c & s & m & a & t & h \\
m & a & t & i & c & s & m & a & t & h & e \\
a & t & i & c & s & m & a & t & h & e & m \\
t & i & c & s & m & a & t & h & e & m & a \\
i & c & s & m & a & t & h & e & m & a & t \\
c & s & m & a & t & h & e & m & a & t & i \\
s & m & a & t & h & e & m & a & t & i & c
\end{array}
$$

$$
\begin{array}{cccccccccccc}
1 & a & t & h & e & m & a & t & i & c & s & m \\
2 & a & t & i & c & s & m & a & t & h & e & m \\
3 & c & s & m & a & t & h & e & m & a & t & i \\
4 & e & m & a & t & i & c & s & m & a & t & h \\
5 & h & e & m & a & t & i & c & s & m & a & t \\
6 & i & c & s & m & a & t & h & e & m & a & t \\
7 & m & a & t & h & e & m & a & t & i & c & s \\
8 & m & a & t & i & c & s & m & a & t & h & e \\
9 & s & m & a & t & h & e & m & a & t & i & c \\
10 & t & h & e & m & a & t & i & c & s & m & a \\
11 & t & i & c & s & m & a & t & h & e & m & a
\end{array}
$$

Each row of M is a conjugate of w in lexicographic order.
The index I is the row of M containing the original word.
$bwt(w) = L = mmihttsecaa$ and $I = 7$.

The Burrows Wheeler Transform

Given a word $w \in \Sigma^*$, $bwt(w)$ is a permutation of w, obtained as concatenation of the last letters of the lexicographically sorted list of its conjugates.

Example: $w = mathematics$

$$\begin{array}{ccccccccccc}
m & a & t & h & e & m & a & t & i & c & s \\
a & t & h & e & m & a & t & i & c & s & m \\
t & h & e & m & a & t & i & c & s & m & a \\
h & e & m & a & t & i & c & s & m & a & t \\
e & m & a & t & i & c & s & m & a & t & h \\
m & a & t & i & c & s & m & a & t & h & e \\
a & t & i & c & s & m & a & t & h & e & m \\
t & i & c & s & m & a & t & h & e & m & a \\
i & c & s & m & a & t & h & e & m & a & t \\
c & s & m & a & t & h & e & m & a & t & i \\
s & m & a & t & h & e & m & a & t & i & c
\end{array}$$

M

$$\begin{array}{cccccccccccc}
1 & a & t & h & e & m & a & t & i & c & s & m \\
2 & a & t & i & c & s & m & a & t & h & e & m \\
3 & c & s & m & a & t & h & e & m & a & t & i \\
4 & e & m & a & t & i & c & s & m & a & t & h \\
5 & h & e & m & a & t & i & c & s & m & a & t \\
6 & i & c & s & m & a & t & h & e & m & a & t \\
7 & m & a & t & h & e & m & a & t & i & c & s \\
8 & m & a & t & i & c & s & m & a & t & h & e \\
9 & s & m & a & t & h & e & m & a & t & i & c \\
10 & t & h & e & m & a & t & i & c & s & m & a \\
11 & t & i & c & s & m & a & t & h & e & m & a
\end{array}$$

Each row of M is a conjugate of w in lexicographic order.

The index I is the row of M containing the original word.

$bwt(w) = L = mmihttsecaa$ and $I = 7$.

The Burrows Wheeler Transform

Given a word $w \in \Sigma^*$, $bwt(w)$ is a permutation of w, obtained as concatenation of the last letters of the lexicographically sorted list of its conjugates.

Example: $w = mathematics$

$$\begin{array}{ccccccccccc}
m & a & t & h & e & m & a & t & i & c & s \\
a & t & h & e & m & a & t & i & c & s & m \\
t & h & e & m & a & t & i & c & s & m & a \\
h & e & m & a & t & i & c & s & m & a & t \\
e & m & a & t & i & c & s & m & a & t & h \\
m & a & t & i & c & s & m & a & t & h & e \\
a & t & i & c & s & m & a & t & h & e & m \\
t & i & c & s & m & a & t & h & e & m & a \\
i & c & s & m & a & t & h & e & m & a & t \\
c & s & m & a & t & h & e & m & a & t & i \\
s & m & a & t & h & e & m & a & t & i & c
\end{array}$$

M

$$\begin{array}{r|ccccccccccc}
1 & a & t & h & e & m & a & t & i & c & s & m \\
2 & a & t & i & c & s & m & a & t & h & e & m \\
3 & c & s & m & a & t & h & e & m & a & t & i \\
4 & e & m & a & t & i & c & s & m & a & t & h \\
5 & h & e & m & a & t & i & c & s & m & a & t \\
6 & i & c & s & m & a & t & h & e & m & a & t \\
I \rightarrow 7 & m & a & t & h & e & m & a & t & i & c & s \\
8 & m & a & t & i & c & s & m & a & t & h & e \\
9 & s & m & a & t & h & e & m & a & t & i & c \\
10 & t & h & e & m & a & t & i & c & s & m & a \\
11 & t & i & c & s & m & a & t & h & e & m & a
\end{array}$$

Each row of M is a conjugate of w in lexicographic order.

The index I is the row of M containing the original word.

$bwt(w) = L = mmihttsecaa$ and $I = 7$.

The Burrows Wheeler Transform

Given a word $w \in \Sigma^*$, $bwt(w)$ is a permutation of w, obtained as concatenation of the last letters of the lexicographically sorted list of its conjugates.

Example: $w = mathematics$

$$
\begin{array}{ccccccccccc}
m & a & t & h & e & m & a & t & i & c & s \\
a & t & h & e & m & a & t & i & c & s & m \\
t & h & e & m & a & t & i & c & s & m & a \\
h & e & m & a & t & i & c & s & m & a & t \\
e & m & a & t & i & c & s & m & a & t & h \\
m & a & t & i & c & s & m & a & t & h & e \\
a & t & i & c & s & m & a & t & h & e & m \\
t & i & c & s & m & a & t & h & e & m & a \\
i & c & s & m & a & t & h & e & m & a & t \\
c & s & m & a & t & h & e & m & a & t & i \\
s & m & a & t & h & e & m & a & t & i & c
\end{array}
$$

$$
M
$$

$$
\begin{array}{rccccccccccc}
 & & & & & & & & & & & L \\
 & & & & & & & & & & & \downarrow \\
1 & a & t & h & e & m & a & t & i & c & s & m \\
2 & a & t & i & c & s & m & a & t & h & e & m \\
3 & c & s & m & a & t & h & e & m & a & t & i \\
4 & e & m & a & t & i & c & s & m & a & t & h \\
5 & h & e & m & a & t & i & c & s & m & a & t \\
6 & i & c & s & m & a & t & h & e & m & a & t \\
I \rightarrow 7 & m & a & t & h & e & m & a & t & i & c & s \\
8 & m & a & t & i & c & s & m & a & t & h & e \\
9 & s & m & a & t & h & e & m & a & t & i & c \\
10 & t & h & e & m & a & t & i & c & s & m & a \\
11 & t & i & c & s & m & a & t & h & e & m & a
\end{array}
$$

Each row of M is a conjugate of w in lexicographic order.
The index I is the row of M containing the original word.
$bwt(w) = L = mmihttsecaa$ and $I = 7$.

The Burrows Wheeler Transform

Given a word $w \in \Sigma^*$, $bwt(w)$ is a permutation of w, obtained as concatenation of the last letters of the lexicographically sorted list of its conjugates.

Example: $w = mathematics$

$$\begin{array}{ccccccccccc}
m & a & t & h & e & m & a & t & i & c & s \\
a & t & h & e & m & a & t & i & c & s & m \\
t & h & e & m & a & t & i & c & s & m & a \\
h & e & m & a & t & i & c & s & m & a & t \\
e & m & a & t & i & c & s & m & a & t & h \\
m & a & t & i & c & s & m & a & t & h & e \\
a & t & i & c & s & m & a & t & h & e & m \\
t & i & c & s & m & a & t & h & e & m & a \\
i & c & s & m & a & t & h & e & m & a & t \\
c & s & m & a & t & h & e & m & a & t & i \\
s & m & a & t & h & e & m & a & t & i & c
\end{array}$$

$$M$$

$$\begin{array}{rccccccccccc}
 & & & & & & & & & & & L \\
 & & & & & & & & & & & \downarrow \\
1 & a & t & h & e & m & a & t & i & c & s & m \\
2 & a & t & i & c & s & m & a & t & h & e & m \\
3 & c & s & m & a & t & h & e & m & a & t & i \\
4 & e & m & a & t & i & c & s & m & a & t & h \\
5 & h & e & m & a & t & i & c & s & m & a & t \\
6 & i & c & s & m & a & t & h & e & m & a & t \\
I \rightarrow 7 & m & a & t & h & e & m & a & t & i & c & s \\
8 & m & a & t & i & c & s & m & a & t & h & e \\
9 & s & m & a & t & h & e & m & a & t & i & c \\
10 & t & h & e & m & a & t & i & c & s & m & a \\
11 & t & i & c & s & m & a & t & h & e & m & a
\end{array}$$

Each row of M is a conjugate of w in lexicographic order.
The index I is the row of M containing the original word.
$bwt(w) = L = mmihttsecaa$ and $I = 7$.

The Burrows Wheeler Transform

Given a word $w \in \Sigma^*$, $bwt(w)$ is a permutation of w, obtained as concatenation of the last letters of the lexicographically sorted list of its conjugates.

Example: $w = mathematics$

$$\begin{array}{ccccccccccc}
m & a & t & h & e & m & a & t & i & c & s \\
a & t & h & e & m & a & t & i & c & s & m \\
t & h & e & m & a & t & i & c & s & m & a \\
h & e & m & a & t & i & c & s & m & a & t \\
e & m & a & t & i & c & s & m & a & t & h \\
m & a & t & i & c & s & m & a & t & h & e \\
a & t & i & c & s & m & a & t & h & e & m \\
t & i & c & s & m & a & t & h & e & m & a \\
i & c & s & m & a & t & h & e & m & a & t \\
c & s & m & a & t & h & e & m & a & t & i \\
s & m & a & t & h & e & m & a & t & i & c
\end{array}$$

$$M$$

$$\begin{array}{rccccccccccc}
 & F & & & & & & & & & & L \\
 & \downarrow & & & & & & & & & & \downarrow \\
1 & a & t & h & e & m & a & t & i & c & s & m \\
2 & a & t & i & c & s & m & a & t & h & e & m \\
3 & c & s & m & a & t & h & e & m & a & t & i \\
4 & e & m & a & t & i & c & s & m & a & t & h \\
5 & h & e & m & a & t & i & c & s & m & a & t \\
6 & i & c & s & m & a & t & h & e & m & a & t \\
I \rightarrow 7 & m & a & t & h & e & m & a & t & i & c & s \\
8 & m & a & t & i & c & s & m & a & t & h & e \\
9 & s & m & a & t & h & e & m & a & t & i & c \\
10 & t & h & e & m & a & t & i & c & s & m & a \\
11 & t & i & c & s & m & a & t & h & e & m & a
\end{array}$$

Each row of M is a conjugate of w in lexicographic order.
The index I is the row of M containing the original word.
$bwt(w) = L = mmihttsecaa$ and $I = 7$.

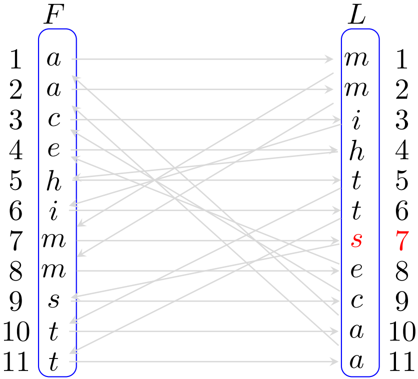
Properties and Reversibility

Example: $bwt(w) = L = mmihttsecaa$ and $I = 7$

- The last letter of w is $L[I]$.
- For each letter z, the i-th occurrence of z in L corresponds to the i-th occurrence of z in F;

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 7 & 8 & 6 & 5 & 10 & 11 & 9 & 4 & 3 & 1 & 2 \end{pmatrix}$$

- For all $i = 1, \ldots, n$ and $i \neq I$, the letter $L[i]$ precedes $F[i]$ in the original word.



$w = \qquad\qquad s$

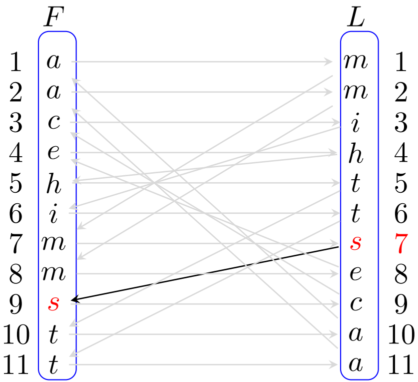
Properties and Reversibility

Example: $bwt(w) = L = mmihttsecaa$ and $I = 7$

- The last letter of w is $L[I]$.
- For each letter z, the i-th occurrence of z in L corresponds to the i-th occurrence of z in F;

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 7 & 8 & 6 & 5 & 10 & 11 & 9 & 4 & 3 & 1 & 2 \end{pmatrix}$$

- For all $i = 1, \ldots, n$ and $i \neq I$, the letter $L[i]$ precedes $F[i]$ in the original word.



$w = \quad s$

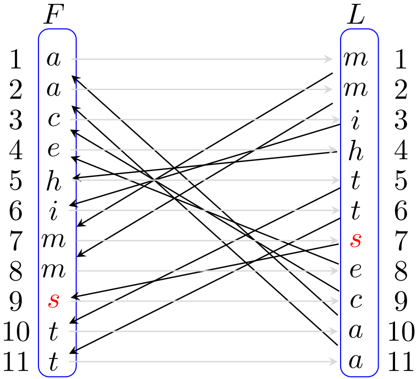
Properties and Reversibility

Example: $bwt(w) = L = mmihttsecaa$ and $I = 7$

- The last letter of w is $L[I]$.
- For each letter z, the i-th occurrence of z in L corresponds to the i-th occurrence of z in F;

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 7 & 8 & 6 & 5 & 10 & 11 & 9 & 4 & 3 & 1 & 2 \end{pmatrix}$$

- For all $i = 1, \ldots, n$ and $i \neq I$, the letter $L[i]$ precedes $F[i]$ in the original word.



$w = \qquad\qquad s$

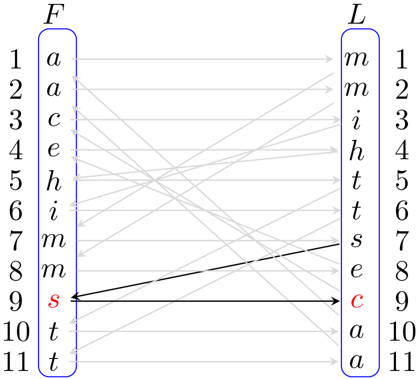
Properties and Reversibility

Example: $bwt(w) = L = mmihttsecaa$ and $I = 7$

- The last letter of w is $L[I]$.
- For each letter z, the i-th occurrence of z in L corresponds to the i-th occurrence of z in F;

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 7 & 8 & 6 & 5 & 10 & 11 & 9 & 4 & 3 & 1 & 2 \end{pmatrix}$$

- For all $i = 1, \ldots, n$ and $i \neq I$, the letter $L[i]$ precedes $F[i]$ in the original word.



$w = \qquad cs$

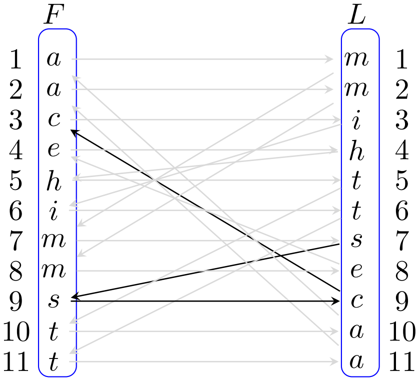
Properties and Reversibility

Example: $bwt(w) = L = mmihttsecaa$ and $I = 7$

- The last letter of w is $L[I]$.
- For each letter z, the i-th occurrence of z in L corresponds to the i-th occurrence of z in F;

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 7 & 8 & 6 & 5 & 10 & 11 & 9 & 4 & 3 & 1 & 2 \end{pmatrix}$$

- For all $i = 1, \ldots, n$ and $i \neq I$, the letter $L[i]$ precedes $F[i]$ in the original word.



$w =$ cs

Properties and Reversibility

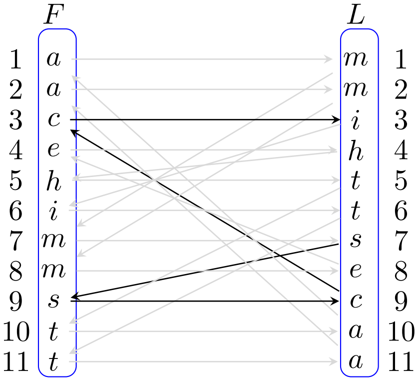
Example: $bwt(w) = L = mmihttsecaa$ and $I = 7$

- The last letter of w is $L[I]$.
- For each letter z, the i-th occurrence of z in L corresponds to the i-th occurrence of z in F;

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 7 & 8 & 6 & 5 & 10 & 11 & 9 & 4 & 3 & 1 & 2 \end{pmatrix}$$

- For all $i = 1, \ldots, n$ and $i \neq I$, the letter $L[i]$ precedes $F[i]$ in the original word.

$w = \qquad ics$



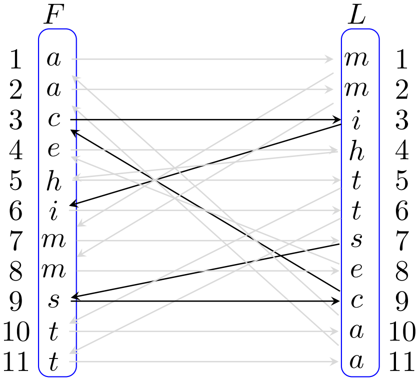
Properties and Reversibility

Example: $bwt(w) = L = mmihttsecaa$ and $I = 7$

- The last letter of w is $L[I]$.
- For each letter z, the i-th occurrence of z in L corresponds to the i-th occurrence of z in F;

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 7 & 8 & 6 & 5 & 10 & 11 & 9 & 4 & 3 & 1 & 2 \end{pmatrix}$$

- For all $i = 1, \ldots, n$ and $i \neq I$, the letter $L[i]$ precedes $F[i]$ in the original word.



$w = \qquad\qquad ics$

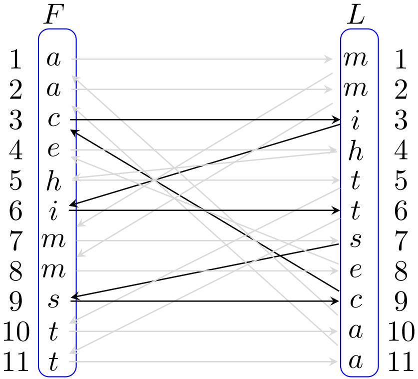
Properties and Reversibility

Example: $bwt(w) = L = mmihttsecaa$ and $I = 7$

- The last letter of w is $L[I]$.
- For each letter z, the i-th occurrence of z in L corresponds to the i-th occurrence of z in F;

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 7 & 8 & 6 & 5 & 10 & 11 & 9 & 4 & 3 & 1 & 2 \end{pmatrix}$$

- For all $i = 1, \ldots, n$ and $i \neq I$, the letter $L[i]$ precedes $F[i]$ in the original word.



$w =$ $tics$

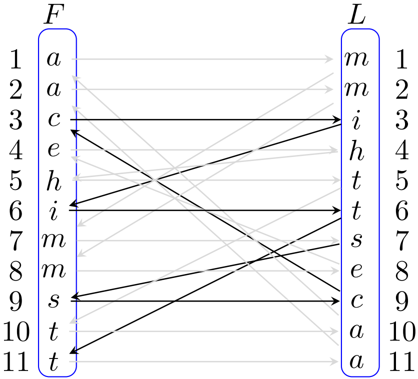
Properties and Reversibility

Example: $bwt(w) = L = mmihttsecaa$ and $I = 7$

- The last letter of w is $L[I]$.
- For each letter z, the i-th occurrence of z in L corresponds to the i-th occurrence of z in F;

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 7 & 8 & 6 & 5 & 10 & 11 & 9 & 4 & 3 & 1 & 2 \end{pmatrix}$$

- For all $i = 1, \ldots, n$ and $i \neq I$, the letter $L[i]$ precedes $F[i]$ in the original word.



$w = \quad tics$

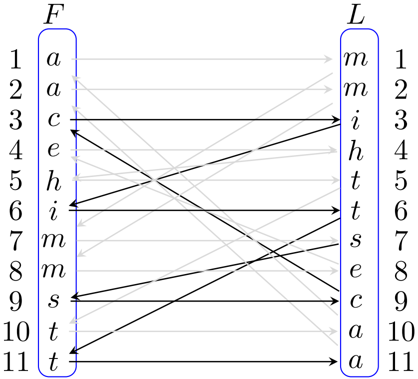
Properties and Reversibility

Example: $bwt(w) = L = mmihttsecaa$ and $I = 7$

- The last letter of w is $L[I]$.
- For each letter z, the i-th occurrence of z in L corresponds to the i-th occurrence of z in F;

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 7 & 8 & 6 & 5 & 10 & 11 & 9 & 4 & 3 & 1 & 2 \end{pmatrix}$$

- For all $i = 1, \ldots, n$ and $i \neq I$, the letter $L[i]$ precedes $F[i]$ in the original word.



$w =$ $atics$

Properties and Reversibility

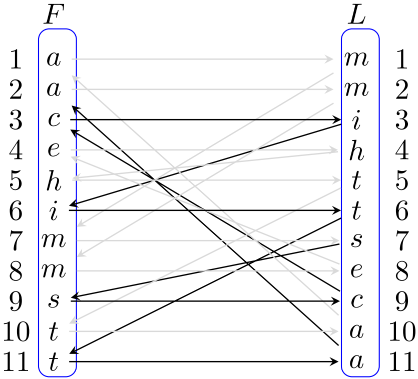
Example: $bwt(w) = L = mmihttsecaa$ and $I = 7$

- The last letter of w is $L[I]$.
- For each letter z, the i-th occurrence of z in L corresponds to the i-th occurrence of z in F;

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 7 & 8 & 6 & 5 & 10 & 11 & 9 & 4 & 3 & 1 & 2 \end{pmatrix}$$

- For all $i = 1, \ldots, n$ and $i \neq I$, the letter $L[i]$ precedes $F[i]$ in the original word.

$w =$ $atics$

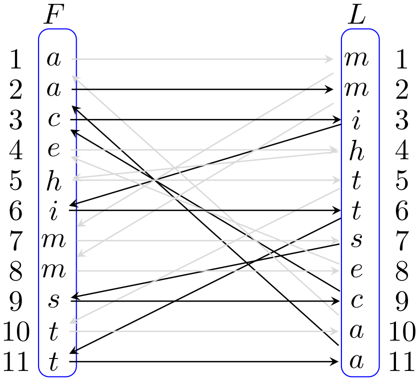


Properties and Reversibility

Example: $bwt(w) = L = mmihttsecaa$ and $I = 7$

- The last letter of w is $L[I]$.
- For each letter z, the i-th occurrence of z in L corresponds to the i-th occurrence of z in F;

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 7 & 8 & 6 & 5 & 10 & 11 & 9 & 4 & 3 & 1 & 2 \end{pmatrix}$$



- For all $i = 1, \ldots, n$ and $i \neq I$, the letter $L[i]$ precedes $F[i]$ in the original word.

$w = \qquad matics$

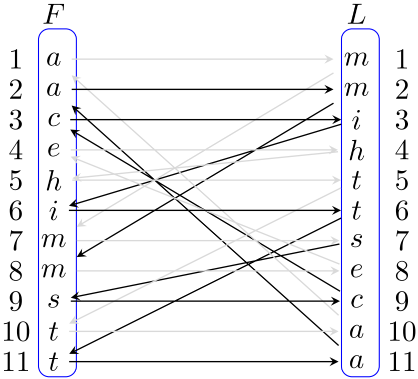
Properties and Reversibility

Example: $bwt(w) = L = mmihttsecaa$ and $I = 7$

- The last letter of w is $L[I]$.
- For each letter z, the i-th occurrence of z in L corresponds to the i-th occurrence of z in F;

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 7 & 8 & 6 & 5 & 10 & 11 & 9 & 4 & 3 & 1 & 2 \end{pmatrix}$$

- For all $i = 1, \dots, n$ and $i \neq I$, the letter $L[i]$ precedes $F[i]$ in the original word.



$w = \quad matics$

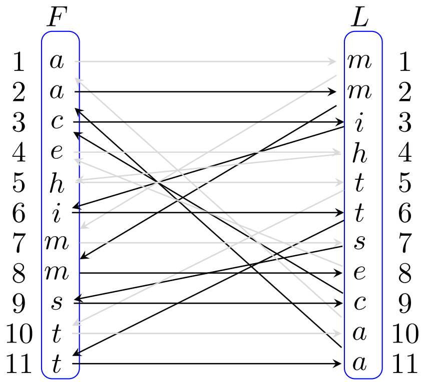
Properties and Reversibility

Example: $bwt(w) = L = mmihttsecaa$ and $I = 7$

- The last letter of w is $L[I]$.
- For each letter z, the i-th occurrence of z in L corresponds to the i-th occurrence of z in F;

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 7 & 8 & 6 & 5 & 10 & 11 & 9 & 4 & 3 & 1 & 2 \end{pmatrix}$$

- For all $i = 1, \ldots, n$ and $i \neq I$, the letter $L[i]$ precedes $F[i]$ in the original word.



$w = \qquad ematics$

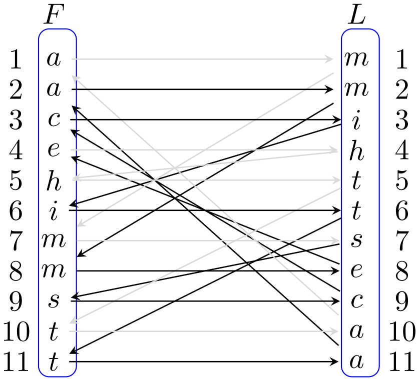
Properties and Reversibility

Example: $bwt(w) = L = mmihttsecaa$ and $I = 7$

- The last letter of w is $L[I]$.
- For each letter z, the i-th occurrence of z in L corresponds to the i-th occurrence of z in F;

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 7 & 8 & 6 & 5 & 10 & 11 & 9 & 4 & 3 & 1 & 2 \end{pmatrix}$$

- For all $i = 1, \ldots, n$ and $i \neq I$, the letter $L[i]$ precedes $F[i]$ in the original word.



$w = \quad ematics$

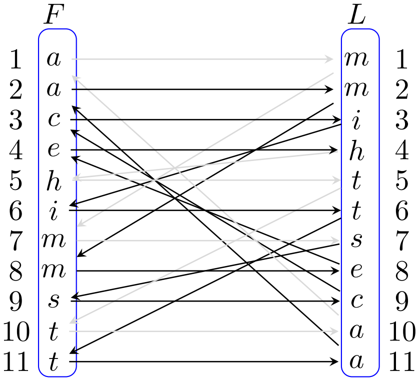
Properties and Reversibility

Example: $bwt(w) = L = mmihttsecaa$ and $I = 7$

- The last letter of w is $L[I]$.
- For each letter z, the i-th occurrence of z in L corresponds to the i-th occurrence of z in F;

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 7 & 8 & 6 & 5 & 10 & 11 & 9 & 4 & 3 & 1 & 2 \end{pmatrix}$$

- For all $i = 1, \ldots, n$ and $i \neq I$, the letter $L[i]$ precedes $F[i]$ in the original word.



$w = \quad hematics$

Properties and Reversibility

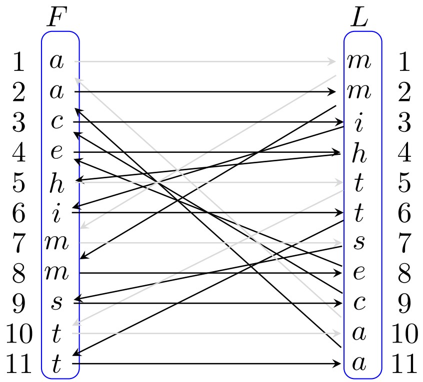
Example: $bwt(w) = L = mmihttsecaa$ and $I = 7$

- The last letter of w is $L[I]$.
- For each letter z, the i-th occurrence of z in L corresponds to the i-th occurrence of z in F;

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 7 & 8 & 6 & 5 & 10 & 11 & 9 & 4 & 3 & 1 & 2 \end{pmatrix}$$

- For all $i = 1, \ldots, n$ and $i \neq I$, the letter $L[i]$ precedes $F[i]$ in the original word.

$w = \quad hematics$



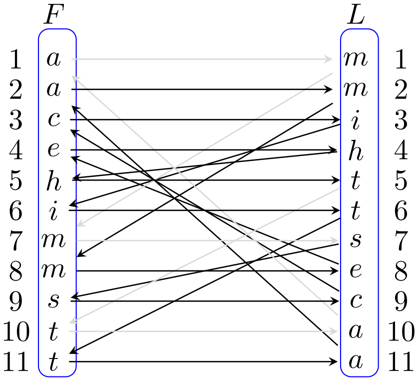
Properties and Reversibility

Example: $bwt(w) = L = mmihttsecaa$ and $I = 7$

- The last letter of w is $L[I]$.
- For each letter z, the i-th occurrence of z in L corresponds to the i-th occurrence of z in F;

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 7 & 8 & 6 & 5 & 10 & 11 & 9 & 4 & 3 & 1 & 2 \end{pmatrix}$$

- For all $i = 1, \ldots, n$ and $i \neq I$, the letter $L[i]$ precedes $F[i]$ in the original word.



$w = \quad thematics$

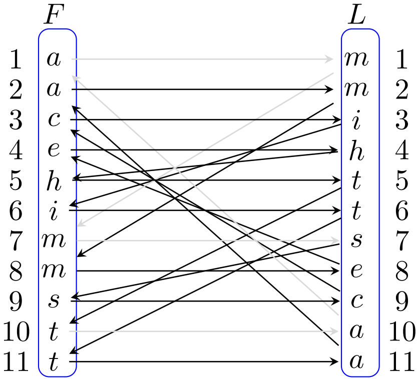
Properties and Reversibility

Example: $bwt(w) = L = mmihttsecaa$ and $I = 7$

- The last letter of w is $L[I]$.
- For each letter z, the i-th occurrence of z in L corresponds to the i-th occurrence of z in F;

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 7 & 8 & 6 & 5 & 10 & 11 & 9 & 4 & 3 & 1 & 2 \end{pmatrix}$$

- For all $i = 1, \ldots, n$ and $i \neq I$, the letter $L[i]$ precedes $F[i]$ in the original word.



$w = \quad thematics$

Properties and Reversibility

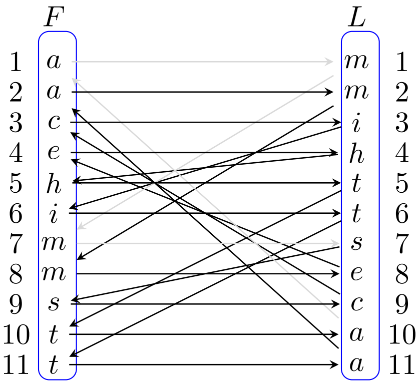
Example: $bwt(w) = L = mmihttsecaa$ and $I = 7$

- The last letter of w is $L[I]$.
- For each letter z, the i-th occurrence of z in L corresponds to the i-th occurrence of z in F;

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 7 & 8 & 6 & 5 & 10 & 11 & 9 & 4 & 3 & 1 & 2 \end{pmatrix}$$

- For all $i = 1, \ldots, n$ and $i \neq I$, the letter $L[i]$ precedes $F[i]$ in the original word.

$w = \quad athematics$



Properties and Reversibility

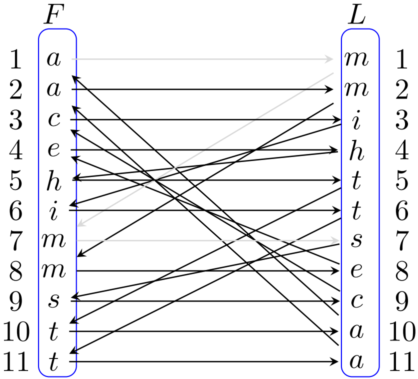
Example: $bwt(w) = L = mmihttsecaa$ and $I = 7$

- The last letter of w is $L[I]$.
- For each letter z, the i-th occurrence of z in L corresponds to the i-th occurrence of z in F;

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 7 & 8 & 6 & 5 & 10 & 11 & 9 & 4 & 3 & 1 & 2 \end{pmatrix}$$

- For all $i = 1, \ldots, n$ and $i \neq I$, the letter $L[i]$ precedes $F[i]$ in the original word.

$w = \quad athematics$



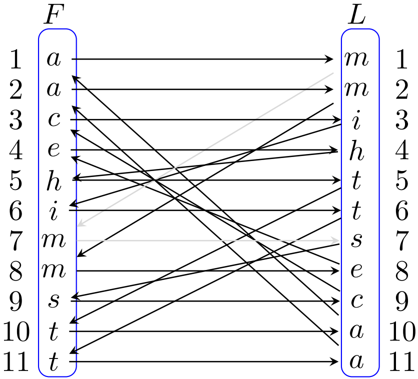
Properties and Reversibility

Example: $bwt(w) = L = mmihttsecaa$ and $I = 7$

- The last letter of w is $L[I]$.
- For each letter z, the i-th occurrence of z in L corresponds to the i-th occurrence of z in F;

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 7 & 8 & 6 & 5 & 10 & 11 & 9 & 4 & 3 & 1 & 2 \end{pmatrix}$$

- For all $i = 1, \ldots, n$ and $i \neq I$, the letter $L[i]$ precedes $F[i]$ in the original word.



$w = mathematics$

Properties and Reversibility

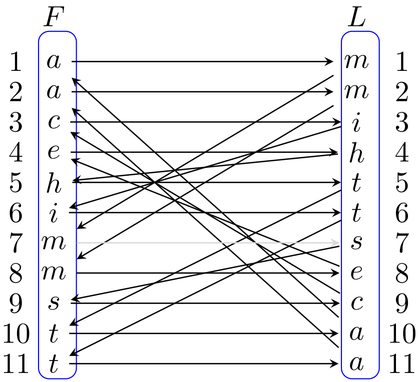
Example: $bwt(w) = L = mmihttsecaa$ and $I = 7$

- The last letter of w is $L[I]$.
- For each letter z, the i-th occurrence of z in L corresponds to the i-th occurrence of z in F;

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 7 & 8 & 6 & 5 & 10 & 11 & 9 & 4 & 3 & 1 & 2 \end{pmatrix}$$

- For all $i = 1, \ldots, n$ and $i \neq I$, the letter $L[i]$ precedes $F[i]$ in the original word.

$w = mathematics$



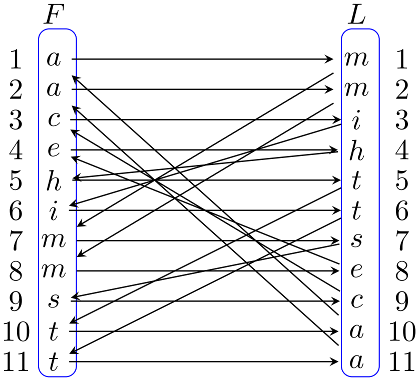
Properties and Reversibility

Example: $bwt(w) = L = mmihttsecaa$ and $I = 7$

- The last letter of w is $L[I]$.
- For each letter z, the i-th occurrence of z in L corresponds to the i-th occurrence of z in F;

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 7 & 8 & 6 & 5 & 10 & 11 & 9 & 4 & 3 & 1 & 2 \end{pmatrix}$$

- For all $i = 1, \ldots, n$ and $i \neq I$, the letter $L[i]$ precedes $F[i]$ in the original word.



$w = mathematics$

Sorting of the conjugates

F	Sorted Conjugates									L
$\downarrow$										$\downarrow$
a	t	h	e	m	a	t	i	c	s	m
a	t	i	c	s	m	a	t	h	e	m
c	s	m	a	t	h	e	m	a	t	i
e	m	a	t	i	c	s	m	a	t	h
h	e	m	a	t	i	c	s	m	a	t
i	c	s	m	a	t	h	e	m	a	t
m	a	t	h	e	m	a	t	i	c	s
m	a	t	i	c	s	m	a	t	h	e
s	m	a	t	h	e	m	a	t	i	c
t	h	e	m	a	t	i	c	s	m	a
t	i	c	s	m	a	t	h	e	m	a

In general, the computation of the sorting of the conjugates of a word is slow!

Sorting the suffixes of a word is a simpler problem. So, in practical applications the sorting of the suffixes is used!

Sorting of the conjugates

F	Sorted Conjugates									L
$\downarrow$										$\downarrow$
a	t	h	e	m	a	t	i	c	s	m
a	t	i	c	s	m	a	t	h	e	m
c	s	m	a	t	h	e	m	a	t	i
e	m	a	t	i	c	s	m	a	t	h
h	e	m	a	t	i	c	s	m	a	t
i	c	s	m	a	t	h	e	m	a	t
m	a	t	h	e	m	a	t	i	c	s
m	a	t	i	c	s	m	a	t	h	e
s	m	a	t	h	e	m	a	t	i	c
t	h	e	m	a	t	i	c	s	m	a
t	i	c	s	m	a	t	h	e	m	a

In general, the computation of the sorting of the conjugates of a word is slow!

Sorting the suffixes of a word is a simpler problem. So, in practical applications the sorting of the suffixes is used!

Sorting of the conjugates

$$
\begin{array}{c}
F \qquad \text{Sorted Conjugates} \qquad L \\
\downarrow \qquad\qquad\qquad\qquad\qquad \downarrow \\
\begin{array}{ccccccccccc}
a & t & h & e & m & a & t & i & c & s & m \\
a & t & i & c & s & m & a & t & h & e & m \\
c & s & m & a & t & h & e & m & a & t & i \\
e & m & a & t & i & c & s & m & a & t & h \\
h & e & m & a & t & i & c & s & m & a & t \\
i & c & s & m & a & t & h & e & m & a & t \\
m & a & t & h & e & m & a & t & i & c & s \\
m & a & t & i & c & s & m & a & t & h & e \\
s & m & a & t & h & e & m & a & t & i & c \\
t & h & e & m & a & t & i & c & s & m & a \\
t & i & c & s & m & a & t & h & e & m & a
\end{array}
\end{array}
$$

In general, the computation of the sorting of the conjugates of a word is slow!

Sorting the suffixes of a word is a simpler problem. So, in practical applications the sorting of the suffixes is used!

Sorting of the suffixes

To ensure the reversibility of the transform, one needs to append the symbol $\$$ at the end of the input string $w \in \Sigma^*$, where $\$ \notin \Sigma = \{a_1, a_2, \ldots, a_k\}$ and $\$ < a_1 < a_2 < \ldots < a_k$.

$bwt(w\$)$ is a permutation of $w\$$, obtained as concatenation of the letters that (circularly) precede the first symbol of the suffix in the list of its lexicographically sorted suffixes.

BWT	Sorted Suffixes
s	$\$$
m	$a\ t\ h\ e\ m\ a\ t\ i\ c\ s\ \$$
m	$a\ t\ i\ c\ s\ \$$
i	$c\ s\ \$$
h	$e\ m\ a\ t\ i\ c\ s\ \$$
t	$h\ e\ m\ a\ t\ i\ c\ s\ \$$
t	$i\ c\ s\ \$$
$\$$	$m\ a\ t\ h\ e\ m\ a\ t\ i\ c\ s\ \$$
e	$m\ a\ t\ i\ c\ s\ \$$
c	$s\ \$$
a	$t\ h\ e\ m\ a\ t\ i\ c\ s\ \$$
a	$t\ i\ c\ s\ \$$

The suffix array and the BWT

Given a word $w \in \Sigma^*$, with $|w| = n$:

- $SA[i]$: The starting position of the ith smallest suffix of $w\$$.
- $BWT[i]$: The symbol that (circularly) precedes the first symbol of the ith smallest suffix.

$$\begin{array}{ccccccccccccc} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ v = & m & a & t & h & e & m & a & t & i & c & s & \$ \end{array}$$

SA	BWT	Sorted Suffixes
12	s	$\$$
2	m	$a\ t\ h\ e\ m\ a\ t\ i\ c\ s\ \$$
7	m	$a\ t\ i\ c\ s\ \$$
9	i	$c\ s\ \$$
5	h	$e\ m\ a\ t\ i\ c\ s\ \$$
4	t	$h\ e\ m\ a\ t\ i\ c\ s\ \$$
9	t	$i\ c\ s\ \$$
1	$\$$	$m\ a\ t\ h\ e\ m\ a\ t\ i\ c\ s\ \$$
6	e	$m\ a\ t\ i\ c\ s\ \$$
11	c	$s\ \$$
3	a	$t\ h\ e\ m\ a\ t\ i\ c\ s\ \$$
8	a	$t\ i\ c\ s\ \$$

More efficient!

By sorting of the suffixes

- There exist several algorithms in time linear for the construction of the SA (see survey of [*Puglisi and Smyth*, 2007]).
- There exist several algorithms in external memory for the construction either of the BWT or of the SA (for instance [*Ferragina, Gagie and Manzini*, 2012]).
- Recently, an in-place computation of the BWT has been proposed in [*Crochemore, Grossi, Kärkkäinen and Landau*, 2013], in which the space occupied by word w is used to store the $bwt(w)$.

By sorting of the suffixes

- There exist several algorithms in time linear for the construction of the SA (see survey of [*Puglisi and Smyth*, 2007]).
- There exist several algorithms in external memory for the construction either of the BWT or of the SA (for instance [*Ferragina, Gagie and Manzini*, 2012]).
- Recently, an in-place computation of the BWT has been proposed in [*Crochemore, Grossi, Kärkkäinen and Landau*, 2013], in which the space occupied by word w is used to store the $bwt(w)$.

By sorting of the suffixes

- There exist several algorithms in time linear for the construction of the SA (see survey of [*Puglisi and Smyth*, 2007]).
- There exist several algorithms in external memory for the construction either of the BWT or of the SA (for instance [*Ferragina, Gagie and Manzini*, 2012]).
- Recently, an in-place computation of the BWT has been proposed in [*Crochemore, Grossi, Kärkkäinen and Landau*, 2013], in which the space occupied by word w is used to store the $bwt(w)$.

Conjugates and Suffixes

In spite of the closeness of these variants, the sorting processes involve different sorting relations on different objects:

- lexicographic order among suffixes of a single word;
- lexicographic order among conjugates of a single word;

Note that,

- in general, the sorting of the conjugates of a word w and the sorting of the suffixes of a word $w\$$ is different.
- for a Lyndon word lexicographic sorting of the suffixes and lexicographic sorting of the conjugates are equivalent. So, one can obtain in linear time the sorting of conjugates of a word by using Lyndon word (cf. *Giancarlo, Restivo and S.*, 2007).

Conjugates and Suffixes

In spite of the closeness of these variants, the sorting processes involve different sorting relations on different objects:

- lexicographic order among suffixes of a single word;
- lexicographic order among conjugates of a single word;

Note that,

- in general, the sorting of the conjugates of a word w and the sorting of the suffixes of a word $w\$$ is different.
- for a Lyndon word lexicographic sorting of the suffixes and lexicographic sorting of the conjugates are equivalent. So, one can obtain in linear time the sorting of conjugates of a word by using Lyndon word (cf. *Giancarlo, Restivo and S.*, 2007).

Conjugates and Suffixes

In spite of the closeness of these variants, the sorting processes involve different sorting relations on different objects:

- lexicographic order among suffixes of a single word;
- lexicographic order among conjugates of a single word;

Note that,

- in general, the sorting of the conjugates of a word w and the sorting of the suffixes of a word $w\$$ is different.
- for a Lyndon word lexicographic sorting of the suffixes and lexicographic sorting of the conjugates are equivalent. So, one can obtain in linear time the sorting of conjugates of a word by using Lyndon word (cf. *Giancarlo, Restivo and S.*, 2007).

Conjugates and Suffixes

In spite of the closeness of these variants, the sorting processes involve different sorting relations on different objects:

- lexicographic order among suffixes of a single word;
- lexicographic order among conjugates of a single word;

Note that,

- in general, the sorting of the conjugates of a word w and the sorting of the suffixes of a word $w\$$ is different.
- for a Lyndon word lexicographic sorting of the suffixes and lexicographic sorting of the conjugates are equivalent. So, one can obtain in linear time the sorting of conjugates of a word by using Lyndon word (cf. *Giancarlo, Restivo and S.*, 2007).

Conjugates and Suffixes

In spite of the closeness of these variants, the sorting processes involve different sorting relations on different objects:

- lexicographic order among suffixes of a single word;
- lexicographic order among conjugates of a single word;

Note that,

- in general, the sorting of the conjugates of a word w and the sorting of the suffixes of a word $w\$$ is different.
- for a Lyndon word lexicographic sorting of the suffixes and lexicographic sorting of the conjugates are equivalent. So, one can obtain in linear time the sorting of conjugates of a word by using Lyndon word (cf. *Giancarlo, Restivo and S.*, 2007).

BWT of a word by its Lyndon Factorization

Theorem (*Chen, Fox and Lyndon*, 1958)

Every word $w \in \Sigma^+$ has a unique factorization $w = w_1 \cdots w_s$ such that $w_1 \geq_{lex} \cdots \geq_{lex} w_s$ is a non-increasing sequence of Lyndon words.

Let $w = abaaaabaaaabaaaabaaaaaab$. The Lyndon factorization of w is

$$ab|aaaab|aaaaabaaaab|aaaaaab$$

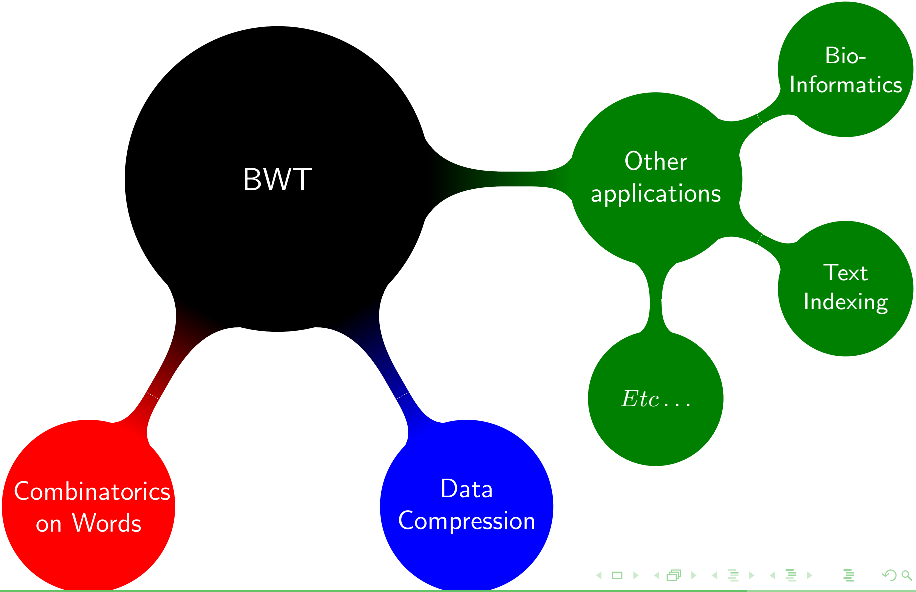
The Lyndon factorization of a given word can be computed in linear time [*Duval* 1983].

Theorem (*Mantaci, Restivo, Rosone and S.*, 2013)

BWT of w can be computed by sorting the suffixes of the Lyndon factors of w.

Property: the local suffixes inside factors keep their mutual order when extended to the suffixes of the whole word.

BWT as tool


BWT
Other applications
Bio-Informatics
Text Indexing
Etc . . .
Combinatorics on Words
Data Compression

Characterization of bwt images

- The word $v = caraab$ is a bwt image, because $bwt(abraca) = caraab$.
- The word $u = bccaaab$ is not a bwt image.
- All words $a^p b^q$ are not bwt images.

Problem

Characterizing all the words in Σ^* that are images by bwt of some word in Σ^*.

Theorem (*Likhomanov and Shur*, 2011)

A characterization of the words that are bwt images is given in terms of combinatorial properties of the permutation π.

Characterization of bwt images

- The word $v = caraab$ is a bwt image, because $bwt(abraca) = caraab$.
- The word $u = bccaaab$ is not a bwt image.
- All words $a^p b^q$ are not bwt images.

Problem

Characterizing all the words in Σ^* that are images by bwt of some word in Σ^*.

Theorem (*Likhomanov and Shur*, 2011)

A characterization of the words that are bwt images is given in terms of combinatorial properties of the permutation π.

Characterization of bwt images

- The word $v = caraab$ is a bwt image, because $bwt(abraca) = caraab$.
- The word $u = bccaaab$ is not a bwt image.
- All words $a^p b^q$ are not bwt images.

Problem

Characterizing all the words in Σ^* that are images by bwt of some word in Σ^*.

Theorem (*Likhomanov and Shur*, 2011)

A characterization of the words that are bwt images is given in terms of combinatorial properties of the permutation π.

Characterization of bwt images

- The word $v = caraab$ is a bwt image, because $bwt(abraca) = caraab$.
- The word $u = bccaaab$ is not a bwt image.
- All words $a^p b^q$ are not bwt images.

Problem

Characterizing all the words in Σ^* that are images by bwt of some word in Σ^*.

Theorem (*Likhomanov and Shur*, 2011)

A characterization of the words that are bwt images is given in terms of combinatorial properties of the permutation π.

Characterization of bwt images

- The word $v = caraab$ is a bwt image, because $bwt(abraca) = caraab$.
- The word $u = bccaaab$ is not a bwt image.
- All words $a^p b^q$ are not bwt images.

Problem

Characterizing all the words in Σ^* that are images by bwt of some word in Σ^*.

Theorem (*Likhomanov and Shur*, 2011)

A characterization of the words that are bwt images is given in terms of combinatorial properties of the permutation π.

Perfectly clustering words

Problem

Characterizing the perfectly clustering words by bwt, that are the words that are transformed by bwt into expressions in which all the occurrences of the same characters are consecutive, such as $c^i b^j a^h$ or $d^i b^j c^h a^k$.

Theorem (*Mantaci, Restivo and S.*, 2003)

Given a word u over the alphabet $\{a, b\}$, $bwt(u) = b^p a^q$ (with $\gcd(p, q) = 1$) if and only if u is a conjugate of a standard sturmian word.

Standard sturmian words

Let $d_1, d_2, \ldots, d_n, \ldots$ be, with $d_1 \geq 0$ and $d_i > 0$ for $i = 2, \ldots, n, \ldots$, the directive sequence, each finite word s_n, where $s_0 = b$, $s_1 = a$, and $s_{n+1} = s_n^{d_n} s_{n-1}$, for $n \geq 1$, is a standard sturmian word.

Simple bwt words

A special attention has been given to the words with *simple* bwt.

Definition

A word w over an ordered alphabet $\Sigma = \{a_1, a_2, \ldots, a_k\}$ with $a_1 < a_2 < \ldots < a_k$, has a simple bwt, if $bwt(w)$ is of the form $a_k^{n_k} a_{k-1}^{n_{k-1}} \cdots a_1^{n_1}$, for some positive integers $n_1, n_2, \ldots, n_k$.

Example

The word $v = acbcbcadad$ is a simple bwt word, in fact $bwt(v) = ddcccbbaaa$.

Simple bwt words: three letters alphabets

Simpson and Puglisi get a constructive characterization of the set of simple bwt words in the case of three letters alphabet.

Theorem (*Simpson and Puglisi*, 2008, *Pak and Redlich*, 2008)

The word u is a primitive word having a simple bwt on the alphabet $\Sigma = \{a_1, a_2, a_3\}$, i.e. $bwt(u) = a_3^{n_3} a_2^{n_2} a_1^{n_1}$, if and only if (n_1, n_2, n_3) is a triple of integers satisfying both the conditions $\gcd(n_1, n_2, n_3) = 1$ and $\gcd(n_1 + n_2, n_2 + n_3) = 1$.

Open problem

This result that involves the vector of the occurrences of the characters cannot be naturally extended for greater alphabets.
The question is still open.

Simple bwt words

Theorem (*Restivo and Rosone*, 2009)

If the word $w \in \Sigma^*$ of length n has a simple bwt then ww has $2n+1$ distinct palindromic factors.

Example

The word $v = acbcbcadad$ is a simple bwt, $|v| = 10$, in fact $bwt(acbcbcadad) = ddcccbbaaa$. The word vv contains 21 distinct palindromic factors.

We note that the converse of this result is false, for instance $bwt(ccaaccb) = cacccba$ and $ccaaccbccaaccb$ has 15 distinct palindromic factors.

Simple bwt words

Theorem (*Restivo and Rosone*, 2009)

If the word $w \in \Sigma^*$ of length n has a simple bwt then ww has $2n+1$ distinct palindromic factors.

Example

The word $v = acbcbcadad$ is a simple bwt, $|v| = 10$, in fact $bwt(acbcbcadad) = ddcccbbaaa$. The word vv contains 21 distinct palindromic factors.

We note that the converse of this result is false, for instance $bwt(ccaaccb) = cacccba$ and $ccaaccbccaaccb$ has 15 distinct palindromic factors.

Perfectly clustering words

In [*Ferenczi and Zamboni*, 2013] it is proved that perfectly clustering words are intrinsically related to k-discrete interval exchange transformations.

Theorem

Perfectly clustering words are exactly those words $w \in \Sigma^*$ such that ww occurs in a trajectory of a k-discrete interval exchange transformation, where k is the size of Σ.

BWT, Clustering effect and Compression

$$v \longrightarrow \boxed{BWT} \longrightarrow bwt(v) \longrightarrow \boxed{\text{Compressor}} \longrightarrow \text{Output}$$

- The application of the BWT produces a clustering effect.
- BWT-based compressors, in general, take advantage of such clustering effect.
- Perfect clustering corresponds to optimal performances of some BWT-based compression algorithms.

Balanced words and compression

What kind of regularity of the input text produces a good compression ratio?

The (experimental) answer:

Balanced input text!

It seems that the output of BWT is more compressible if the input is very close to be balanced.

Is there a statistic that allows to decide whether a text is more compressible by using the BWT?

The (experimental) answer:

Local Entropy of the input text!

The notion of local entropy seems to be a measure of the degree of balance of a text.

Conjecture

The more balanced the input word is, the more local similarity one has after BWT, and the better the compression is.

Intuitively a word is *circularly balanced* if for any pair of different factors of the same length, the frequency of the characters of the alphabet is almost the same.

Balanced words and compression

What kind of regularity of the input text produces a good compression ratio?
The (experimental) answer:
Balanced input text!
It seems that the output of BWT is more compressible if the input is very close to be balanced.

Is there a statistic that allows to decide whether a text is more compressible by using the BWT?
The (experimental) answer:
Local Entropy of the input text!
The notion of local entropy seems to be a measure of the degree of balance of a text.

Conjecture

The more balanced the input word is, the more local similarity one has after BWT, and the better the compression is.

Intuitively a word is *circularly balanced* if for any pair of different factors of the same length, the frequency of the characters of the alphabet is almost the same.

Balanced words and compression

What kind of regularity of the input text produces a good compression ratio?

The (experimental) answer:

Balanced input text!

It seems that the output of BWT is more compressible if the input is very close to be balanced.

Is there a statistic that allows to decide whether a text is more compressible by using the BWT?

The (experimental) answer:

Local Entropy of the input text!

The notion of local entropy seems to be a measure of the degree of balance of a text.

Conjecture

The more balanced the input word is, the more local similarity one has after BWT, and the better the compression is.

Intuitively a word is *circularly balanced* if for any pair of different factors of the same length, the frequency of the characters of the alphabet is almost the same.

Balanced words and compression

What kind of regularity of the input text produces a good compression ratio?
The (experimental) answer:
Balanced input text!
It seems that the output of BWT is more compressible if the input is very close to be balanced.

Is there a statistic that allows to decide whether a text is more compressible by using the BWT?
The (experimental) answer:
Local Entropy of the input text!
The notion of local entropy seems to be a measure of the degree of balance of a text.

Conjecture

The more balanced the input word is, the more local similarity one has after BWT, and the better the compression is.

Intuitively a word is *circularly balanced* if for any pair of different factors of the same length, the frequency of the characters of the alphabet is almost the same.

Balanced words and compression

What kind of regularity of the input text produces a good compression ratio?
The (experimental) answer:
Balanced input text!
It seems that the output of BWT is more compressible if the input is very close to be balanced.

Is there a statistic that allows to decide whether a text is more compressible by using the BWT?
The (experimental) answer:
Local Entropy of the input text!
The notion of local entropy seems to be a measure of the degree of balance of a text.

Conjecture

The more balanced the input word is, the more local similarity one has after BWT, and the better the compression is.

Intuitively a word is *circularly balanced* if for any pair of different factors of the same length, the frequency of the characters of the alphabet is almost the same.

Statistic: Local Entropy based on Distance Coding

Distance coding: for each symbol of the input word, the DC algorithm outputs the distance to the previous occurrence of the same symbol (in circular way).

Example

$$\begin{array}{rccccccc} v = & a & c & b & c & a & a & b \\ dc(v) = & & & & & & & \end{array}$$

Let $v = b_1 b_2 \cdots b_n$, $b_i \in A$ and $dc(v) = d_1 d_2 \cdots d_n$, where $0 \le d_i < n$. Define the Local Entropy of v:

$$LE(v) = \frac{1}{n} \sum_{i=1}^{n} \log(d_i + 1)$$

Local entropy (LE) has been considered by

- *Bentley, Sleator, Tarjan and Wei*, 1986
- *Manzini*, 2001
- *Kaplan, Landau and Verbin*, 2007

Statistic: Local Entropy based on Distance Coding

Distance coding: for each symbol of the input word, the DC algorithm outputs the distance to the previous occurrence of the same symbol (in circular way).

Example

$$\begin{array}{rccccccc} v = & a & c & b & c & a & a & b \\ dc(v) = & 1 & & & & & & \end{array}$$

Let $v = b_1 b_2 \cdots b_n$, $b_i \in A$ and $dc(v) = d_1 d_2 \cdots d_n$, where $0 \leq d_i < n$.
Define the Local Entropy of v:

$$LE(v) = \frac{1}{n} \sum_{i=1}^{n} \log(d_i + 1)$$

Local entropy (LE) has been considered by

- *Bentley, Sleator, Tarjan and Wei*, 1986
- *Manzini*, 2001
- *Kaplan, Landau and Verbin*, 2007

Statistic: Local Entropy based on Distance Coding

Distance coding: for each symbol of the input word, the DC algorithm outputs the distance to the previous occurrence of the same symbol (in circular way).

Example

$$\begin{array}{rccccccc} v = & a & c & b & c & a & a & b \\ dc(v) = & 1 & 4 & & & & & \end{array}$$

Let $v = b_1 b_2 \cdots b_n$, $b_i \in A$ and $dc(v) = d_1 d_2 \cdots d_n$, where $0 \leq d_i < n$. Define the Local Entropy of v:

$$LE(v) = \frac{1}{n} \sum_{i=1}^{n} \log(d_i + 1)$$

Local entropy (LE) has been considered by

- *Bentley, Sleator, Tarjan and Wei*, 1986
- *Manzini*, 2001
- *Kaplan, Landau and Verbin*, 2007

Statistic: Local Entropy based on Distance Coding

Distance coding: for each symbol of the input word, the DC algorithm outputs the distance to the previous occurrence of the same symbol (in circular way).

Example

$$\begin{array}{rccccccc} v = & a & c & b & c & a & a & b \\ dc(v) = & 1 & 4 & 2 & & & & \end{array}$$

Let $v = b_1 b_2 \cdots b_n$, $b_i \in A$ and $dc(v) = d_1 d_2 \cdots d_n$, where $0 \leq d_i < n$. Define the Local Entropy of v:

$$LE(v) = \frac{1}{n} \sum_{i=1}^{n} \log(d_i + 1)$$

Local entropy (LE) has been considered by

- *Bentley, Sleator, Tarjan and Wei*, 1986
- *Manzini*, 2001
- *Kaplan, Landau and Verbin*, 2007

Statistic: Local Entropy based on Distance Coding

Distance coding: for each symbol of the input word, the DC algorithm outputs the distance to the previous occurrence of the same symbol (in circular way).

Example

$$\begin{array}{rccccccc} v = & a & c & b & c & a & a & b \\ dc(v) = & 1 & 4 & 2 & 1 & & & \end{array}$$

Let $v = b_1 b_2 \cdots b_n$, $b_i \in A$ and $dc(v) = d_1 d_2 \cdots d_n$, where $0 \leq d_i < n$.
Define the Local Entropy of v:

$$LE(v) = \frac{1}{n} \sum_{i=1}^{n} \log(d_i + 1)$$

Local entropy (LE) has been considered by

- *Bentley, Sleator, Tarjan and Wei*, 1986
- *Manzini*, 2001
- *Kaplan, Landau and Verbin*, 2007

Statistic: Local Entropy based on Distance Coding

Distance coding: for each symbol of the input word, the DC algorithm outputs the distance to the previous occurrence of the same symbol (in circular way).

Example

$$\begin{array}{rccccccc} v = & a & c & b & c & a & a & b \\ dc(v) = & 1 & 4 & 2 & 1 & 3 & & \end{array}$$

Let $v = b_1 b_2 \cdots b_n$, $b_i \in A$ and $dc(v) = d_1 d_2 \cdots d_n$, where $0 \leq d_i < n$.
Define the Local Entropy of v:

$$LE(v) = \frac{1}{n} \sum_{i=1}^{n} \log(d_i + 1)$$

Local entropy (LE) has been considered by

- *Bentley, Sleator, Tarjan and Wei*, 1986
- *Manzini*, 2001
- *Kaplan, Landau and Verbin*, 2007

Statistic: Local Entropy based on Distance Coding

Distance coding: for each symbol of the input word, the DC algorithm outputs the distance to the previous occurrence of the same symbol (in circular way).

Example

$$\begin{array}{rccccccc} v = & a & c & b & c & a & a & b \\ dc(v) = & 1 & 4 & 2 & 1 & 3 & 0 & \end{array}$$

Let $v = b_1 b_2 \cdots b_n$, $b_i \in A$ and $dc(v) = d_1 d_2 \cdots d_n$, where $0 \leq d_i < n$. Define the Local Entropy of v:

$$LE(v) = \frac{1}{n} \sum_{i=1}^{n} \log(d_i + 1)$$

Local entropy (LE) has been considered by

- *Bentley, Sleator, Tarjan and Wei*, 1986
- *Manzini*, 2001
- *Kaplan, Landau and Verbin*, 2007

Statistic: Local Entropy based on Distance Coding

Distance coding: for each symbol of the input word, the DC algorithm outputs the distance to the previous occurrence of the same symbol (in circular way).

Example

$$\begin{array}{rccccccc} v = & a & c & b & c & a & a & b \\ dc(v) = & 1 & 4 & 2 & 1 & 3 & 0 & 3 \end{array}$$

Let $v = b_1 b_2 \cdots b_n$, $b_i \in A$ and $dc(v) = d_1 d_2 \cdots d_n$, where $0 \leq d_i < n$. Define the Local Entropy of v:

$$LE(v) = \frac{1}{n} \sum_{i=1}^{n} \log(d_i + 1)$$

Local entropy (LE) has been considered by

- *Bentley, Sleator, Tarjan and Wei*, 1986
- *Manzini*, 2001
- *Kaplan, Landau and Verbin*, 2007

Statistic: Local Entropy based on Distance Coding

Distance coding: for each symbol of the input word, the DC algorithm outputs the distance to the previous occurrence of the same symbol (in circular way).

Example

$$\begin{array}{rccccccc} v = & a & c & b & c & a & a & b \\ dc(v) = & 1 & 4 & 2 & 1 & 3 & 0 & 3 \end{array}$$

Let $v = b_1 b_2 \cdots b_n$, $b_i \in A$ and $dc(v) = d_1 d_2 \cdots d_n$, where $0 \leq d_i < n$. Define the Local Entropy of v:

$$LE(v) = \frac{1}{n} \sum_{i=1}^{n} \log(d_i + 1)$$

Local entropy (LE) has been considered by

- *Bentley, Sleator, Tarjan and Wei*, 1986
- *Manzini*, 2001
- *Kaplan, Landau and Verbin*, 2007

Bounds

Theorem (*Restivo and Rosone*, 2011)

For any word v one has:

- $G(v) \leq LE(v) \leq H_0(v)$
- $LE(v) = H_0(v)$ *if and only if v is a constant gap word.*
- $LE(v) = G(v)$ *if and only if v is a clustered word.*

where
$H_0(v) = \sum_{a \in A} \frac{|v|_a}{|v|} \log \frac{|v|}{|v|_a}$, *and* $G(v) = \sum_{a \in A} \frac{1}{|v|}[\log(|v| - |v|_a + 1)]$.

The notion of local entropy can be used in order to define a measure of the degree of balance of a text.

Constant gap words

A finite word v is *constant gap* if, for each letter a, the distance (the number of letters) between two consecutive occurrences of a is constant (in circular way).
$|v|_a$ denotes the number of occurrences of the letter a in the word v.

Preliminary experiments

File name	Size	H_0	Bst	Gzip	Diff %	$\delta(v)$	$\tau(bwt(v))$
bible	4,047,392	4.343	796,231	1,191,071	9.755	0.117	0.233
english	52,428,800	4.529	11,533,171	19,672,355	15.524	0.136	0.238
etext99	105,277,340	4.596	24,949,871	39,493,346	13.814	0.141	0.264
english	104,857,600	4.556	23,993,810	39,437,704	14.728	0.143	0.250
dblp.xml	52,428,800	5.230	4,871,450	9,034,902	7.941	0.152	0.093
dblp.xml	296,135,874	5.262	25,597,003	50,481,103	8.403	0.164	0.086
world192	2,473,400	4.998	430,225	724,606	11.902	0.174	0.183
rctail96	114,711,151	5.154	11,429,406	24,007,508	10.965	0.178	0.097
sprot34.dat	109,617,186	4.762	18,850,472	26,712,981	7.173	0.215	0.206
jdk13c	69,728,899	5.531	3,187,900	7,525,172	6.220	0.224	0.041
howto	39,886,973	4.857	8,713,851	12,638,334	9.839	0.231	0.229
rfc	116,421,901	4.623	17,565,908	26,712,981	7.857	0.239	0.163
w3c2	104,201,579	5.954	7,021,478	15,159,804	7.810	0.246	0.058
chr22.dna	34,553,758	2.137	8,015,707	8,870,068	2.473	0.341	0.575
pitches	52,428,800	5.633	18,651,999	16,884,651	-3.371	0.530	0.344
pitches	55,832,855	5.628	19,475,065	16,040,370	-6.152	0.533	0.337

- $\delta(v)$ measures the degree of balancing of the input text v;
- $\tau(bwt(v))$ measures the degree of clustering of the $bwt(v)$.

The experiments show that when $\delta(v)$ is less than 0.23, then $\tau(bwt(v))$ is less than 0.3 and the BWT-based compressor (bst) has good performances.

Practical application: the computation of $\delta(v)$ is a fast test for the choice between bst and gzip.

Preliminary experiments

File name	Size	H_0	Bst	Gzip	Diff %	$\delta(v)$	$\tau(bwt(v))$
bible	4,047,392	4.343	796,231	1,191,071	9.755	0.117	0.233
english	52,428,800	4.529	11,533,171	19,672,355	15.524	0.136	0.238
etext99	105,277,340	4.596	24,949,871	39,493,346	13.814	0.141	0.264
english	104,857,600	4.556	23,993,810	39,437,704	14.728	0.143	0.250
dblp.xml	52,428,800	5.230	4,871,450	9,034,902	7.941	0.152	0.093
dblp.xml	296,135,874	5.262	25,597,003	50,481,103	8.403	0.164	0.086
world192	2,473,400	4.998	430,225	724,606	11.902	0.174	0.183
rctail96	114,711,151	5.154	11,429,406	24,007,508	10.965	0.178	0.097
sprot34.dat	109,617,186	4.762	18,850,472	26,712,981	7.173	0.215	0.206
jdk13c	69,728,899	5.531	3,187,900	7,525,172	6.220	0.224	0.041
howto	39,886,973	4.857	8,713,851	12,638,334	9.839	0.231	0.229
rfc	116,421,901	4.623	17,565,908	26,712,981	7.857	0.239	0.163
w3c2	104,201,579	5.954	7,021,478	15,159,804	7.810	0.246	0.058
chr22.dna	34,553,758	2.137	8,015,707	8,870,068	2.473	0.341	0.575
pitches	52,428,800	5.633	18,651,999	16,884,651	-3.371	0.530	0.344
pitches	55,832,855	5.628	19,475,065	16,040,370	-6.152	0.533	0.337

- $\delta(v)$ measures the degree of balancing of the input text v;
- $\tau(bwt(v))$ measures the degree of clustering of the $bwt(v)$.

The experiments show that when $\delta(v)$ is less than 0.23, then $\tau(bwt(v))$ is less than 0.3 and the BWT-based compressor (bst) has good performances.

Practical application: the computation of $\delta(v)$ is a fast test for the choice between bst and gzip.

Preliminary experiments

File name	Size	H_0	Bst	Gzip	Diff %	$\delta(v)$	$\tau(bwt(v))$
bible	4,047,392	4.343	796,231	1,191,071	9.755	0.117	0.233
english	52,428,800	4.529	11,533,171	19,672,355	15.524	0.136	0.238
etext99	105,277,340	4.596	24,949,871	39,493,346	13.814	0.141	0.264
english	104,857,600	4.556	23,993,810	39,437,704	14.728	0.143	0.250
dblp.xml	52,428,800	5.230	4,871,450	9,034,902	7.941	0.152	0.093
dblp.xml	296,135,874	5.262	25,597,003	50,481,103	8.403	0.164	0.086
world192	2,473,400	4.998	430,225	724,606	11.902	0.174	0.183
rctail96	114,711,151	5.154	11,429,406	24,007,508	10.965	0.178	0.097
sprot34.dat	109,617,186	4.762	18,850,472	26,712,981	7.173	0.215	0.206
jdk13c	69,728,899	5.531	3,187,900	7,525,172	6.220	0.224	0.041
howto	39,886,973	4.857	8,713,851	12,638,334	9.839	0.231	0.229
rfc	116,421,901	4.623	17,565,908	26,712,981	7.857	0.239	0.163
w3c2	104,201,579	5.954	7,021,478	15,159,804	7.810	0.246	0.058
chr22.dna	34,553,758	2.137	8,015,707	8,870,068	2.473	0.341	0.575
pitches	52,428,800	5.633	18,651,999	16,884,651	-3.371	0.530	0.344
pitches	55,832,855	5.628	19,475,065	16,040,370	-6.152	0.533	0.337

- $\delta(v)$ measures the degree of balancing of the input text v;
- $\tau(bwt(v))$ measures the degree of clustering of the $bwt(v)$.

The experiments show that when $\delta(v)$ is less than 0.23, then $\tau(bwt(v))$ is less than 0.3 and the BWT-based compressor (bst) has good performances.

Practical application: the computation of $\delta(v)$ is a fast test for the choice between bst and gzip.

Multiset of words

Problem

Is it possible to extend the notion of BWT to a multiset of words?

The Extended Burrows-Wheeler Transform [*Mantaci, Restivo, Rosone and S.*, 2005]

Given two words $u, v \in \Sigma^*$ we define the following order relation:

$$u \preceq_\omega v \Longleftrightarrow u^\omega <_{lex} v^\omega$$

where $u^\omega = uuuuu\cdots$ and $v^\omega = vvvvv\cdots$.

Given a set of words , $S = \{w_1, w_2, \ldots w_m\}$ with $w_1, w_2, \ldots w_m \in \Sigma^*$ the EBWT is a transformation that produces a word obtained by sorting according to the $\preceq_\omega$ order the conjugates of the words in S and by taking the concatenation of the last letters of the sorted list.

The Extended Burrows-Wheeler Transform

Consider the set $S = \{abac, bca, cbab, cba\}$.

- Sort all the conjugates of the words in S by the $\preceq_\omega$ order relation;
- Consider the list of the sorted words and take the word L obtained by concatenating the last letter of each word;
- Take the set $\mathcal{I}$ containing the positions of the words corresponding to the ones in S;

$$
\begin{array}{l}
a\ b\ a\ c\ a\ b\ \cdots \\
a\ b\ c\ a\ b\ c\ \cdots \\
a\ b\ c\ b\ a\ b\ \cdots \\
a\ c\ a\ b\ a\ c\ \cdots \\
a\ c\ b\ a\ c\ b\ \cdots \\
b\ a\ b\ c\ b\ a\ \cdots \\
b\ a\ c\ a\ b\ a\ \cdots \\
b\ a\ c\ b\ a\ c\ \cdots \\
b\ c\ a\ b\ c\ a\ \cdots \\
b\ c\ b\ a\ b\ c\ \cdots \\
c\ a\ b\ a\ c\ a\ \cdots \\
c\ a\ b\ c\ a\ b\ \cdots \\
c\ b\ a\ b\ c\ b\ \cdots \\
c\ b\ a\ c\ b\ a\ \cdots
\end{array}
\Longrightarrow
\begin{array}{rl}
1 & a\ b\ a\ \mathbf{c} \\
2 & a\ b\ \mathbf{c} \\
3 & a\ b\ c\ \mathbf{b} \\
4 & a\ c\ a\ \mathbf{b} \\
5 & a\ c\ \mathbf{b} \\
6 & b\ a\ b\ \mathbf{c} \\
7 & b\ a\ c\ \mathbf{a} \\
8 & b\ a\ \mathbf{c} \\
9 & b\ c\ \mathbf{a} \\
10 & b\ c\ b\ \mathbf{a} \\
11 & c\ a\ b\ \mathbf{a} \\
12 & c\ a\ \mathbf{b} \\
13 & c\ b\ a\ \mathbf{b} \\
14 & c\ b\ \mathbf{a}
\end{array}
$$

The output of $ebwt(S)$ is the couple $(L, \mathcal{I})$ where $L = ccbbbcacaaabba$ and $\mathcal{I} = \{1, 9, 13, 14\}$.

The Extended Burrows-Wheeler Transform

Consider the set $S = \{abac, bca, cbab, cba\}$.

- Sort all the conjugates of the words in S by the $\preceq_\omega$ order relation;
- Consider the list of the sorted words and take the word L obtained by concatenating the last letter of each word;
- Take the set $\mathcal{I}$ containing the positions of the words corresponding to the ones in S;

$a\ b\ a\ c\ a\ b \cdots$		1	$a\ b\ a$ **c**
$a\ b\ c\ a\ b\ c \cdots$		2	$a\ b$ **c**
$a\ b\ c\ b\ a\ b \cdots$		3	$a\ b\ c$ **b**
$a\ c\ a\ b\ a\ c \cdots$		4	$a\ c\ a$ **b**
$a\ c\ b\ a\ c\ b \cdots$		5	$a\ c$ **b**
$b\ a\ b\ c\ b\ a \cdots$		6	$b\ a\ b$ **c**
$b\ a\ c\ a\ b\ a \cdots$	$\Longrightarrow$	7	$b\ a\ c$ **a**
$b\ a\ c\ b\ a\ c \cdots$		8	$b\ a$ **c**
$b\ c\ a\ b\ c\ a \cdots$		9	$b\ c$ **a**
$b\ c\ b\ a\ b\ c \cdots$		10	$b\ c\ b$ **a**
$c\ a\ b\ a\ c\ a \cdots$		11	$c\ a\ b$ **a**
$c\ a\ b\ c\ a\ b \cdots$		12	$c\ a$ **b**
$c\ b\ a\ b\ c\ b \cdots$		13	$c\ b\ a$ **b**
$c\ b\ a\ c\ b\ a \cdots$		14	$c\ b$ **a**

The output of $ebwt(S)$ is the couple $(L, \mathcal{I})$ where $L = ccbbbcacaaabba$ and $\mathcal{I} = \{1, 9, 13, 14\}$

The Extended Burrows-Wheeler Transform

Consider the set $S = \{abac, bca, cbab, cba\}$.

- Sort all the conjugates of the words in S by the $\preceq_\omega$ order relation;
- Consider the list of the sorted words and take the word L obtained by concatenating the last letter of each word;
- Take the set $\mathcal{I}$ containing the positions of the words corresponding to the ones in S;

$a\ b\ a\ c\ a\ b \cdots$		1	$a\ b\ a\ \mathbf{c}$
$a\ b\ c\ a\ b\ c \cdots$		2	$a\ b\ \mathbf{c}$
$a\ b\ c\ b\ a\ b \cdots$		3	$a\ b\ c\ \mathbf{b}$
$a\ c\ a\ b\ a\ c \cdots$		4	$a\ c\ a\ \mathbf{b}$
$a\ c\ b\ a\ c\ b \cdots$		5	$a\ c\ \mathbf{b}$
$b\ a\ b\ c\ b\ a \cdots$		6	$b\ a\ b\ \mathbf{c}$
$b\ a\ c\ a\ b\ a \cdots$		7	$b\ a\ c\ \mathbf{a}$
$b\ a\ c\ b\ a\ c \cdots$	$\Longrightarrow$	8	$b\ a\ \mathbf{c}$
$b\ c\ a\ b\ c\ a \cdots$		9	$b\ c\ \mathbf{a}$
$b\ c\ b\ a\ b\ c \cdots$		10	$b\ c\ b\ \mathbf{a}$
$c\ a\ b\ a\ c\ a \cdots$		11	$c\ a\ b\ \mathbf{a}$
$c\ a\ b\ c\ a\ b \cdots$		12	$c\ a\ \mathbf{b}$
$c\ b\ a\ b\ c\ b \cdots$		13	$c\ b\ a\ \mathbf{b}$
$c\ b\ a\ c\ b\ a \cdots$		14	$c\ b\ \mathbf{a}$

The output of $ebwt(S)$ is the couple $(L, \mathcal{I})$ where $L = ccbbbcacaaabba$ and $\mathcal{I} = \{1, 9, 13, 14\}$.

The Extended Burrows-Wheeler Transform

Consider the set $S = \{abac, bca, cbab, cba\}$.

- Sort all the conjugates of the words in S by the $\preceq_\omega$ order relation;
- Consider the list of the sorted words and take the word L obtained by concatenating the last letter of each word;
- Take the set $\mathcal{I}$ containing the positions of the words corresponding to the ones in S;

$a\ b\ a\ c\ a\ b \cdots$	1	$a\ b\ a\ \mathbf{c}$
$a\ b\ c\ a\ b\ c \cdots$	2	$a\ b\ \mathbf{c}$
$a\ b\ c\ b\ a\ b \cdots$	3	$a\ b\ c\ \mathbf{b}$
$a\ c\ a\ b\ a\ c \cdots$	4	$a\ c\ a\ \mathbf{b}$
$a\ c\ b\ a\ c\ b \cdots$	5	$a\ c\ \mathbf{b}$
$b\ a\ b\ c\ b\ a \cdots$	6	$b\ a\ b\ \mathbf{c}$
$b\ a\ c\ a\ b\ a \cdots$ $\Longrightarrow$	7	$b\ a\ c\ \mathbf{a}$
$b\ a\ c\ b\ a\ c \cdots$	8	$b\ a\ \mathbf{c}$
$b\ c\ a\ b\ c\ a \cdots$	9	$b\ c\ \mathbf{a}$
$b\ c\ b\ a\ b\ c \cdots$	10	$b\ c\ b\ \mathbf{a}$
$c\ a\ b\ a\ c\ a \cdots$	11	$c\ a\ b\ \mathbf{a}$
$c\ a\ b\ c\ a\ b \cdots$	12	$c\ a\ \mathbf{b}$
$c\ b\ a\ b\ c\ b \cdots$	13	$c\ b\ a\ \mathbf{b}$
$c\ b\ a\ c\ b\ a \cdots$	14	$c\ b\ \mathbf{a}$

The output of $ebwt(S)$ is the couple $(L, \mathcal{I})$ where $L = ccbbbcacaaabba$ and $\mathcal{I} = \{1, 9, 13, 14\}$.

The Extended Burrows-Wheeler Transform

Consider the set $S = \{abac, bca, cbab, cba\}$.

- Sort all the conjugates of the words in S by the $\preceq_\omega$ order relation;
- Consider the list of the sorted words and take the word L obtained by concatenating the last letter of each word;
- Take the set $\mathcal{I}$ containing the positions of the words corresponding to the ones in S;

$a\ b\ a\ c\ a\ b \cdots$		1	$a\ b\ a\ \mathbf{c}$
$a\ b\ c\ a\ b\ c \cdots$		2	$a\ b\ \mathbf{c}$
$a\ b\ c\ b\ a\ b \cdots$		3	$a\ b\ c\ \mathbf{b}$
$a\ c\ a\ b\ a\ c \cdots$		4	$a\ c\ a\ \mathbf{b}$
$a\ c\ b\ a\ c\ b \cdots$		5	$a\ c\ \mathbf{b}$
$b\ a\ b\ c\ b\ a \cdots$		6	$b\ a\ b\ \mathbf{c}$
$b\ a\ c\ a\ b\ a \cdots$		7	$b\ a\ c\ \mathbf{a}$
$b\ a\ c\ b\ a\ c \cdots$	$\Longrightarrow$	8	$b\ a\ \mathbf{c}$
$b\ c\ a\ b\ c\ a \cdots$		9	$b\ c\ \mathbf{a}$
$b\ c\ b\ a\ b\ c \cdots$		10	$b\ c\ b\ \mathbf{a}$
$c\ a\ b\ a\ c\ a \cdots$		11	$c\ a\ b\ \mathbf{a}$
$c\ a\ b\ c\ a\ b \cdots$		12	$c\ a\ \mathbf{b}$
$c\ b\ a\ b\ c\ b \cdots$		13	$c\ b\ a\ \mathbf{b}$
$c\ b\ a\ c\ b\ a \cdots$		14	$c\ b\ \mathbf{a}$

The output of $ebwt(S)$ is the couple $(L, \mathcal{I})$ where $L = ccbbbcacaaabba$ and $\mathcal{I} = \{1, 9, 13, 14\}$.

Properties and Reversibility

Example: $L = ccbbbcacaaabba$ and $\mathcal{I} = \{1, 9, 13, 14\}$.

- The last character of each word w_j is $L[I_j]$;
- For each character z, the i-th occurrence of z in L corresponds to the i-th occurrence of z in F;
- In any row $i \neq \mathcal{I}$, the character $F[i]$ follows $L[i]$ in a word in S.

1	$a\ b\ a$ **c**
2	$a\ b$ **c**
3	$a\ b\ c$ **b**
4	$a\ c\ a$ **b**
5	$a\ c$ **b**
6	$b\ a\ b$ **c**
7	$b\ a\ c$ **a**
8	$b\ a$ **c**
9	$b\ c$ **a**
10	$b\ c\ b$ **a**
11	$c\ a\ b$ **a**
12	$c\ a$ **b**
13	$c\ b\ a$ **b**
14	$c\ b$ **a**

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 \\ 11 & 12 & 6 & 7 & 8 & 13 & 1 & 14 & 2 & 3 & 4 & 9 & 10 & 5 \end{pmatrix} = (\ 11\ 4\ 7\ 1\ )(\ 9\ 2\ 12\ )(\ 13\ 10\ 3\ 6\ )(\ 14\ 5\ 8\ )$$

So, we can recover each word of the multiset

$$S = \{abac, bca, cbab, cba\}.$$

Properties and Reversibility

Example: $L = ccbbbcacaaabba$ and $\mathcal{I} = \{1, 9, 13, 14\}$.

- The last character of each word w_j is $L[I_j]$;
- For each character z, the i-th occurrence of z in L corresponds to the i-th occurrence of z in F;
- In any row $i \neq \mathcal{I}$, the character $F[i]$ follows $L[i]$ in a word in S.

1	a	b	a	**c**	
2	a	b	**c**		
3	a	b	c	**b**	
4	a	c	a	**b**	
5	a	c	**b**		
6	b	a	b	**c**	
7	b	a	c	**a**	
8	b	a	**c**		
9	b	c	**a**		
10	b	c	b	**a**	
11	c	a	b	**a**	
12	c	a	**b**		
13	c	b	a	**b**	
14	c	b	**a**		

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 \\ 11 & 12 & 6 & 7 & 8 & 13 & 1 & 14 & 2 & 3 & 4 & 9 & 10 & 5 \end{pmatrix} = (\ 11\ 4\ 7\ 1\ )(\ 9\ 2\ 12\ )(\ 13\ 10\ 3\ 6\ )(\ 14\ 5\ 8\ )$$

So, we can recover each word of the multiset

$$S = \{abac, bca, cbab, cba\}.$$

Properties and Reversibility

Example: $L = ccbbbcacaaabba$ and $\mathcal{I} = \{1, 9, 13, 14\}$.

- The last character of each word w_j is $L[I_j]$;
- For each character z, the i-th occurrence of z in L corresponds to the i-th occurrence of z in F;
- In any row $i \neq \mathcal{I}$, the character $F[i]$ follows $L[i]$ in a word in S.

1	a	b	a	**c**	
2	a	b	**c**		
3	a	b	c	**b**	
4	a	c	a	**b**	
5	a	c	**b**		
6	b	a	b	**c**	
7	b	a	c	**a**	
8	b	a	**c**		
9	b	c	**a**		
10	b	c	b	**a**	
11	c	a	b	**a**	
12	c	a	**b**		
13	c	b	a	**b**	
14	c	b	**a**		

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 \\ 11 & 12 & 6 & 7 & 8 & 13 & 1 & 14 & 2 & 3 & 4 & 9 & 10 & 5 \end{pmatrix} = (\,11\ 4\ 7\ 1\,)(\,9\ 2\ 12\,)(\,13\ 10\ 3\ 6\,)(\,14\ 5\ 8\,)$$

So, we can recover each word of the multiset

$$S = \{abac, bca, cbab, cba\}.$$

Properties and Reversibility

Example: $L = ccbbbcacaaabba$ and $\mathcal{I} = \{1, 9, 13, 14\}$.

- The last character of each word w_j is $L[I_j]$;
- For each character z, the i-th occurrence of z in L corresponds to the i-th occurrence of z in F;
- In any row $i \neq \mathcal{I}$, the character $F[i]$ follows $L[i]$ in a word in S.

1	$a\ b\ a$ **c**
2	$a\ b$ **c**
3	$a\ b\ c$ **b**
4	$a\ c\ a$ **b**
5	$a\ c$ **b**
6	$b\ a\ b$ **c**
7	$b\ a\ c$ **a**
8	$b\ a$ **c**
9	$b\ c$ **a**
10	$b\ c\ b$ **a**
11	$c\ a\ b$ **a**
12	$c\ a$ **b**
13	$c\ b\ a$ **b**
14	$c\ b$ **a**

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 \\ 11 & 12 & 6 & 7 & 8 & 13 & 1 & 14 & 2 & 3 & 4 & 9 & 10 & 5 \end{pmatrix} = (\ 11\ 4\ 7\ 1\ )(\ 9\ 2\ 12\ )(\ 13\ 10\ 3\ 6\ )(\ 14\ 5\ 8\ )$$

So, we can recover each word of the multiset

$$S = \{abac, bca, cbab, cba\}.$$

Properties and Reversibility

Example: $L = ccbbbcacaaabba$ and $\mathcal{I} = \{1, 9, 13, 14\}$.

- The last character of each word w_j is $L[I_j]$;
- For each character z, the i-th occurrence of z in L corresponds to the i-th occurrence of z in F;
- In any row $i \neq \mathcal{I}$, the character $F[i]$ follows $L[i]$ in a word in S.

1	a	b	a	**c**
2	a	b	**c**	
3	a	b	c	**b**
4	a	c	a	**b**
5	a	c	**b**	
6	b	a	b	**c**
7	b	a	c	**a**
8	b	a	**c**	
9	b	c	**a**	
10	b	c	b	**a**
11	c	a	b	**a**
12	c	a	**b**	
13	c	b	a	**b**
14	c	b	**a**	

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 \\ 11 & 12 & 6 & 7 & 8 & 13 & 1 & 14 & 2 & 3 & 4 & 9 & 10 & 5 \end{pmatrix} = (\ 11\ 4\ 7\ 1\ )(\ 9\ 2\ 12\ )(\ 13\ 10\ 3\ 6\ )(\ 14\ 5\ 8\ )$$

So, we can recover each word of the multiset

$$S = \{abac, bca, cbab, cba\}.$$

EBWT as bijection

Let M be the family of multisets of conjugacy classes of primitive words of Σ^*. Then, if we don't care about the indices $EBWT : M \longrightarrow \Sigma^*$

- The transformation $EBWT$ is injective.
- The $EBWT$ is surjective. For each word $u \in \Sigma^*$, there exists a multiset $S \in M$ such that $EBWT(S) = u$. For instance, $EBWT(ab, abcac) = (bccaaab)$.

Theorem (*Gessel and Reutenauer*, 1993. *Mantaci, Restivo, Rosone and S.*, 2007)

There exists a bijection between Σ^* and the family of multisets of conjugacy classes of primitive words in Σ^*.

EBWT as bijection

Let M be the family of multisets of conjugacy classes of primitive words of Σ^*. Then, if we don't care about the indices $EBWT : M \longrightarrow \Sigma^*$

- The transformation $EBWT$ is injective.
- The $EBWT$ is surjective. For each word $u \in \Sigma^*$, there exists a multiset $S \in M$ such that $EBWT(S) = u$. For instance, $EBWT(ab, abcac) = (bccaaab)$.

Theorem (*Gessel and Reutenauer*, 1993. *Mantaci, Restivo, Rosone and S.*, 2007)

There exists a bijection between Σ^* and the family of multisets of conjugacy classes of primitive words in Σ^*.

EBWT as bijection

Let M be the family of multisets of conjugacy classes of primitive words of Σ^*. Then, if we don't care about the indices $EBWT : M \longrightarrow \Sigma^*$

- The transformation $EBWT$ is injective.
- The $EBWT$ is surjective. For each word $u \in \Sigma^*$, there exists a multiset $S \in M$ such that $EBWT(S) = u$. For instance, $EBWT(ab, abcac) = (bccaaab)$.

Theorem (*Gessel and Reutenauer*, 1993. *Mantaci, Restivo, Rosone and S.*, 2007)

There exists a bijection between Σ^* and the family of multisets of conjugacy classes of primitive words in Σ^*.

Sorting of the conjugates

1	a	b	a	**c**
2	a	b	**c**	
3	a	b	c	**b**
4	a	c	a	**b**
5	a	c	**b**	
6	b	a	b	**c**
7	b	a	c	**a**
8	b	a	**c**	
9	b	c	**a**	
10	b	c	b	**a**
11	c	a	b	**a**
12	c	a	**b**	
13	c	b	a	**b**
14	c	b	**a**	

Sorting the conjugates of each word of the multiset in according to $\preceq_\omega$ order is the bottleneck of the algorithm.

- *Mantaci, Restivo, Rosone and S.*: An extension of the Burrows-Wheeler Transform, 2007. Use a periodicity theorem to reduce the number of comparisons.
- *Hon, Ku, Lu, Shah and Thankachan*: Efficient Algorithm for Circular Burrows-Wheeler Transform, 2011. A $O(n \log n)$ algorithm is provided, where n denotes the total length of the words in S.

Efficient strategy by sorting the suffixes

To ensure the reversibility of the transform, one needs to append a different end-marker at the end of each input string of the multiset.

Let S' be the set of the strings of S included the end-markers. $ebwt(S')$ is a permutation of S', obtained as concatenation of the letters that (circularly) precede the first symbol of the suffix in the list of its lexicographically sorted suffixes of S'.

- *Bauer, Cox and Rosone*: Lightweight algorithms for constructing and inverting the BWT of string collections. 2013.

Efficient strategy by sorting the suffixes

To ensure the reversibility of the transform, one needs to append a different end-marker at the end of each input string of the multiset.

Let S' be the set of the strings of S included the end-markers. $ebwt(S')$ is a permutation of S', obtained as concatenation of the letters that (circularly) precede the first symbol of the suffix in the list of its lexicographically sorted suffixes of S'.

- *Bauer, Cox and Rosone*: Lightweight algorithms for constructing and inverting the BWT of string collections. 2013.

Different sorting relations

In order to compute EBWT of a multiset of words, different sorting processes can be involved.

- Lexicographic order among suffixes of a multiset of words;
- $\preceq_\omega$ order among conjugates of a multiset of words.

A study of the combinatorial aspects that connect these two sorting can be found in *Bonomo, Mantaci, Restivo, Rosone and S.* 2013.
An important role is played by the notion of Lyndon word.

Different sorting relations

In order to compute EBWT of a multiset of words, different sorting processes can be involved.

- Lexicographic order among suffixes of a multiset of words;
- $\preceq_\omega$ order among conjugates of a multiset of words.

A study of the combinatorial aspects that connect these two sorting can be found in *Bonomo, Mantaci, Restivo, Rosone and S.* 2013.
An important role is played by the notion of Lyndon word.

Different sorting relations

In order to compute EBWT of a multiset of words, different sorting processes can be involved.

- Lexicographic order among suffixes of a multiset of words;
- $\preceq_\omega$ order among conjugates of a multiset of words.

A study of the combinatorial aspects that connect these two sorting can be found in *Bonomo, Mantaci, Restivo, Rosone and S.* 2013.
An important role is played by the notion of Lyndon word.

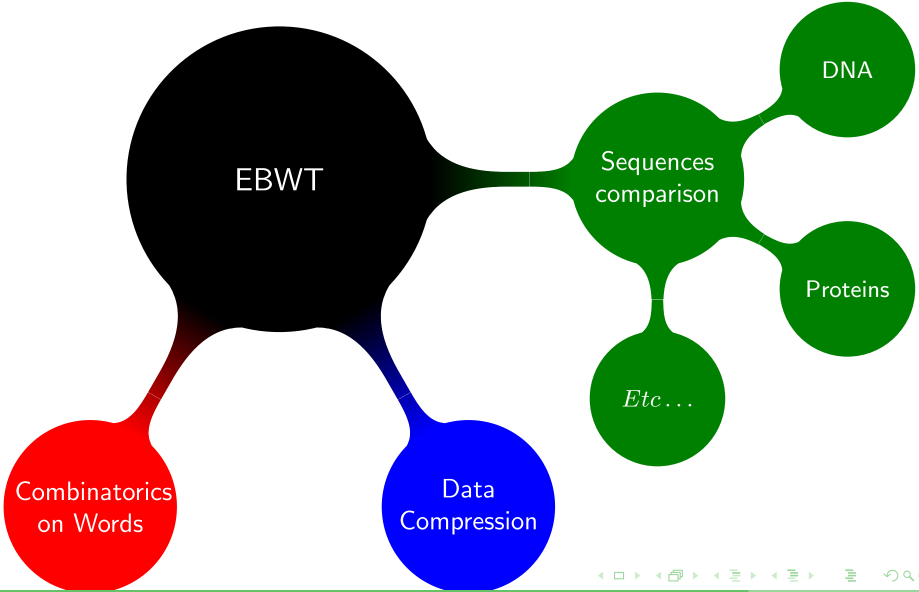
Different sorting relations

In order to compute EBWT of a multiset of words, different sorting processes can be involved.

- Lexicographic order among suffixes of a multiset of words;
- $\preceq_\omega$ order among conjugates of a multiset of words.

A study of the combinatorial aspects that connect these two sorting can be found in *Bonomo, Mantaci, Restivo, Rosone and S.* 2013.
An important role is played by the notion of Lyndon word.

EBWT as tool


EBWT
Sequences comparison
DNA
Proteins
Etc...
Combinatorics on Words
Data Compression

Sequences comparison

The transformation EBWT is used in order to define an alignment-free method for comparing sequences.

The comparison method based on transformation EBWT measures how similar u and v are, by taking into account how much their conjugates are mixed.

Different possible formalizations of distance measures

For instance,

- by computing the number of the alternations in the sequence of colors [*Mantaci, Restivo, Rosone and S.*, 2007].
- by using different partitioning of the colored output of the $EBWT$ and by finally counting the difference of frequencies of colors into each block of the partition [*Mantaci, Restivo, Rosone and S.*, 2008].

For instance, let $S = \{u = ababccb, v = ababccc\}$, the output colored is $bcbbcaaaacccbb$.

Sorted conjugates	EBWT
$ababccb$	b
$ababccc$	c
$abccbab$	b
$abcccab$	b
$bababcc$	c
$babccba$	a
$babccca$	a
$bccbaba$	a
$bcccaba$	a
$cababcc$	c
$cbababc$	c
$ccababc$	c
$ccbabab$	b
$cccabab$	b

Different possible formalizations of distance measures

For instance,

- by computing the number of the alternations in the sequence of colors [*Mantaci, Restivo, Rosone and S.*, 2007].
- by using different partitioning of the colored output of the $EBWT$ and by finally counting the difference of frequencies of colors into each block of the partition [*Mantaci, Restivo, Rosone and S.*, 2008].

For instance, let $S = \{u = ababccb, v = ababccc\}$, the output colored is $bcbbcaaaacccbb$.

Sorted conjugates	EBWT	$\delta(u,v)$
$ababccb$	b	0
$ababccc$	c	0
$abccbab$	b	0
$abcccab$	b	0
$bababcc$	c	
$babccba$	a	1
$babccca$	a	0
$bccbaba$	a	0
$bcccaba$	a	
$cababcc$	c	1
$cbababc$	c	0
$ccababc$	c	0
$ccbabab$	b	0
$cccabab$	b	0

For instance, the number of the alternations in the sequence of colors, computed as:

$$\delta(u,v) = \sum_{\substack{i=1,\\ n_i \neq 0}}^{k} (n_i - 1),$$

is equal to 2.

Different possible formalizations of distance measures

For instance,

- by computing the number of the alternations in the sequence of colors [*Mantaci, Restivo, Rosone and S.*, 2007].
- by using different partitioning of the colored output of the $EBWT$ and by finally counting the difference of frequencies of colors into each block of the partition [*Mantaci, Restivo, Rosone and S.*, 2008].

For instance, let $S = \{u = ababccb, v = ababccc\}$, the output colored is $bcbbcaaaacccbb$.

Sorted conjugates	EBWT
$ababccb$	b
$ababccc$	c
$abccbab$	b
$abcccab$	b
$bababcc$	c
$babccba$	a
$babccca$	a
$bccbaba$	a
$bcccaba$	a
$cababcc$	c
$cbababc$	c
$ccababc$	c
$ccbabab$	b
$cccabab$	b

Different possible formalizations of distance measures

For instance,

- by computing the number of the alternations in the sequence of colors [*Mantaci, Restivo, Rosone and S.*, 2007].
- by using different partitioning of the colored output of the $EBWT$ and by finally counting the difference of frequencies of colors into each block of the partition [*Mantaci, Restivo, Rosone and S.*, 2008].

For instance, let $S = \{u = ababccb, v = ababccc\}$, the output colored is $bcbbcaaaacccbb$.

Sorted conjugates	EBWT	$\varrho(u,v)$
$ababccb$	**b**	1
$ababccc$	c	1
$abccbab$	**b**	0
$abcccab$	**b**	
$bababcc$	c	1
$babccba$	**a**	0
$babccca$	**a**	
$bccbaba$	**a**	
$bcccaba$	**a**	
$cababcc$	c	1
$cbababc$	c	
$ccababc$	c	
$ccbabab$	**b**	0
$cccabab$	**b**	

$$\varrho(u,v) = \sum_{i=1}^{k} |c_i(u) - c_i(v)| = 4$$

Applications to biological sequences

Such distances have been successfully used in several biological datasets, as for instance mitochondrial DNA genomes, expressed sequence tags and proteins

- *Mantaci, Restivo, Rosone and S.*: A New Combinatorial Approach to Sequence Comparison, 2008.
- *Yang, Chang, Zhang and Wang*: Use of the Burrows-Wheeler similarity distribution to the comparison of the proteins, 2010.
- *Yang, Zhang and Wang*: The Burrows-Wheeler similarity distribution between biological sequences based on Burrows-Wheeler transform, 2010.
- *Cox, Jakobi, Rosone and Schulz-Trieglaff*: Comparing DNA Sequence Collections by Direct Comparison of Compressed Text Indexes, 2012.
- *Ng, Ho, and Phon-Amnuaisuk*: A hybrid distance measure for clustering expressed sequence tags originating from the same gene family, 2012.

Massive Datasets

The EBWT has been used as a preprocessing for compression of big sets of m texts.

- *Cox, Bauer, Jakobi and Rosone*: Large-scale compression of genomic sequence databases with the Burrows-Wheeler transform, 2012.
- *Janin, Rosone and Cox*: Adaptive reference-free compression of sequence quality scores, 2013.

The method is also used for the computation of the LCP of very large collections of sequences.

- *Cox, Bauer, Rosone and S.*: Lightweight LCP Construction for Next-Generation Sequencing Datasets, 2012. The code is available as part of the BEETL Library - http://beetl.github.com/BEETL/

Further works and open problems

- Use EBWT to define lightweight data structures for indexing big datasets of sequences;
- Study of the clustering effect of the EBWT from Combinatorics on Words viewpoint.

Thanks for your attention!