UNIVERSITÀ DI PISA

Compression based on Multi-string BWT

Giovanna Rosone

University of Pisa, Italy

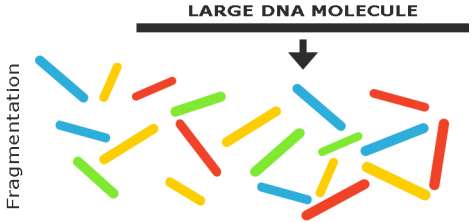
17th Workshop on Compression, Text, and Algorithms

Concepción-Chile, November 11th, 2022

Common thread

Next-generation DNA sequencing

The advent of "next-generation" DNA sequencing (NGS) technologies has meant that very large collections of DNA sequences are commonplace and their compression is always more important.



$\longrightarrow$ Compression

NGS compression - FASTA and FastQ formats

headers

bases

```
>NC_060943.1-123414700
GAGAAAACATACAAGTATATCTTCATGGCCTTAGATTTGGAAGGTGTTTC
```

```
@HWI-ST928:79:C0GNWACXX:6:1101:1278:2193
GCACATGTGTGTTGTTTCAAACGGCAATTTTGGCCAAAATACGGTGTTTTCCGAACAC
+
?@@DDDD=C,A+<EF;CDHHEFGCGFFFIIEFFBEBBCCFBBFFBFFIII>@F>DFFF
```

quality scores

Compression

Lossless
does not lose any data during compression

Lossy
permanently eliminates some information

Headers
- By exploiting structure and high redundancy

Bases
- Reference-based
- Reference-free

Quality score
- Read-based
- not using biological information

Quality score is an integer (character in ASCII) that expresses error probability on the Phred scale $Q_{phred} = -10\log_{10}p$ where p is the error probability.

This talk

Describe strategies for the compression of sequences (FASTA or FASTQ files) of very large collections that exploit the properties of the Burrows-Wheeler Transform:

Bases: reference-free (not relying on external information):

- lossless in terms of bases;
- lossy in terms of input order of the strings in the collection.

Quality scores: (lossy) smooth of quality scores (read-based, i.e. using biological information)

Bases and quality scores: modifying *both components*, bases and quality scores, at the same time (reference-free and read-based):

- lossy in terms of bases;
- lossy in terms of quality scores.

The Burrows-Wheeler Transform (BWT)

The Burrows-Wheeler Transform is a reversible transformation that takes as input a string v and produces:

- a permutation $bwt(v)$ of the symbols of v, obtained as concatenation of the last symbols of the lexicographically sorted list of its cyclic rotations.
- the index I is the position in the sorted list containing the original string.

Example: $v = mathematics$.

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

Output: $bwt(v) = L = mmihttsecaa$ and $I = 7$.

The Burrows-Wheeler Transform (BWT)

The Burrows-Wheeler Transform is a reversible transformation that takes as input a string v and produces:

- a permutation $bwt(v)$ of the symbols of v, obtained as concatenation of the last symbols of the lexicographically sorted list of its cyclic rotations.
- the index I is the position in the sorted list containing the original string.

Example: $v = mathematics$.

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

Output: $bwt(v) = L = mmihttsecaa$ and $I = 7$.

The Burrows-Wheeler Transform (BWT)

The Burrows-Wheeler Transform is a reversible transformation that takes as input a string v and produces:

- a permutation $bwt(v)$ of the symbols of v, obtained as concatenation of the last symbols of the lexicographically sorted list of its cyclic rotations.
- the index I is the position in the sorted list containing the original string.

Example: $v = mathematics$.

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

Output: $bwt(v) = L = mmihttsecaa$ and $I = 7$.

The Burrows-Wheeler Transform (BWT)

The Burrows-Wheeler Transform is a reversible transformation that takes as input a string v and produces:

- a permutation $bwt(v)$ of the symbols of v, obtained as concatenation of the last symbols of the lexicographically sorted list of its cyclic rotations.
- the index I is the position in the sorted list containing the original string.

Example: $v = mathematics$.

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

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

Output: $bwt(v) = L = mmihttsecaa$ and $I = 7$.

The Burrows-Wheeler Transform (BWT)

The Burrows-Wheeler Transform is a reversible transformation that takes as input a string v and produces:

- a permutation $bwt(v)$ of the symbols of v, obtained as concatenation of the last symbols of the lexicographically sorted list of its cyclic rotations.
- the index I is the position in the sorted list containing the original string.

Example: $v = mathematics$.

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	

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

Output: $bwt(v) = L = mmihttsecaa$ and $I = 7$.

The Burrows-Wheeler Transform (BWT)

The Burrows-Wheeler Transform is a reversible transformation that takes as input a string v and produces:

- a permutation $bwt(v)$ of the symbols of v, obtained as concatenation of the last symbols of the lexicographically sorted list of its cyclic rotations.
- the index I is the position in the sorted list containing the original string.

Example: $v = mathematics$.

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

											L ↓
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

Output: $bwt(v) = L = mmihttsecaa$ and $I = 7$.

The Burrows-Wheeler Transform (BWT)

The Burrows-Wheeler Transform is a reversible transformation that takes as input a string v and produces:

- a permutation $bwt(v)$ of the symbols of v, obtained as concatenation of the last symbols of the lexicographically sorted list of its cyclic rotations.
- the index I is the position in the sorted list containing the original string.

Example: $v = mathematics$.

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

											L ↓
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

Output: $bwt(v) = L = mmihttsecaa$ and $I = 7$.

The (extended) Burrows-Wheeler Transform

The Extended Burrows-Wheeler Transform (eBWT)

1. define a new order relation (called ω-order) on the cyclic rotations [Mantaci, Restivo, R. and Sciortino, 2007];
2. append end-markers to each string and use the lexicographic order on the suffixes (called multi-string BWT or eBWT).

eBWT properties

- the strings belonging to S are not concatenated;
- reversible transformation (that produces a permutation of the symbols of the input string collection)
- produces a clustering effect (reduces the number of runs);
- strings can be added/removed (dynamic BWT);
- reconstruction of the entire collection or an its subset.

The (extended) Burrows-Wheeler Transform

The Extended Burrows-Wheeler Transform (eBWT)

1. define a new order relation (called ω-order) on the cyclic rotations [Mantaci, Restivo, R. and Sciortino, 2007];
2. append end-markers to each string and use the lexicographic order on the suffixes (called multi-string BWT or eBWT).

eBWT properties

- the strings belonging to S are not concatenated;
- reversible transformation (that produces a permutation of the symbols of the input string collection)
- produces a clustering effect (reduces the number of runs);
- strings can be added/removed (dynamic BWT);
- reconstruction of the entire collection or an its subset.

The (extended) Burrows-Wheeler Transform

The Extended Burrows-Wheeler Transform (eBWT)

1. define a new order relation (called ω-order) on the cyclic rotations [Mantaci, Restivo, R. and Sciortino, 2007];
2. append end-markers to each string and use the lexicographic order on the suffixes (called multi-string BWT or eBWT).

eBWT properties

- the strings belonging to S are not concatenated;
- reversible transformation (that produces a permutation of the symbols of the input string collection)
- produces a clustering effect (reduces the number of runs);
- strings can be added/removed (dynamic BWT);
- reconstruction of the entire collection or an its subset.

The (extended) Burrows-Wheeler Transform

The Extended Burrows-Wheeler Transform (eBWT)

1. define a new order relation (called ω-order) on the cyclic rotations [Mantaci, Restivo, R. and Sciortino, 2007];
2. append end-markers to each string and use the lexicographic order on the suffixes (called multi-string BWT or eBWT).

eBWT properties

- the strings belonging to S are not concatenated;
- reversible transformation (that produces a permutation of the symbols of the input string collection)
- produces a clustering effect (reduces the number of runs);
- strings can be added/removed (dynamic BWT);
- reconstruction of the entire collection or an its subset.

The (extended) Burrows-Wheeler Transform

The Extended Burrows-Wheeler Transform (eBWT)

1. define a new order relation (called ω-order) on the cyclic rotations [Mantaci, Restivo, R. and Sciortino, 2007];
2. append end-markers to each string and use the lexicographic order on the suffixes (called multi-string BWT or eBWT).

eBWT properties

- the strings belonging to S are not concatenated;
- reversible transformation (that produces a permutation of the symbols of the input string collection)
- produces a clustering effect (reduces the number of runs);
- strings can be added/removed (dynamic BWT);
- reconstruction of the entire collection or an its subset.

The (extended) Burrows-Wheeler Transform

The Extended Burrows-Wheeler Transform (eBWT)

1. define a new order relation (called ω-order) on the cyclic rotations [Mantaci, Restivo, R. and Sciortino, 2007];
2. append end-markers to each string and use the lexicographic order on the suffixes (called multi-string BWT or eBWT).

eBWT properties

- the strings belonging to S are not concatenated;
- reversible transformation (that produces a permutation of the symbols of the input string collection)
- produces a clustering effect (reduces the number of runs);
- strings can be added/removed (dynamic BWT);
- reconstruction of the entire collection or an its subset.

The (extended) Burrows-Wheeler Transform

The Extended Burrows-Wheeler Transform (eBWT)

1. define a new order relation (called ω-order) on the cyclic rotations [Mantaci, Restivo, R. and Sciortino, 2007];
2. append end-markers to each string and use the lexicographic order on the suffixes (called multi-string BWT or eBWT).

eBWT properties

- the strings belonging to S are not concatenated;
- reversible transformation (that produces a permutation of the symbols of the input string collection)
- produces a clustering effect (reduces the number of runs);
- strings can be added/removed (dynamic BWT);
- reconstruction of the entire collection or an its subset.

The (extended) Burrows-Wheeler Transform

The Extended Burrows-Wheeler Transform (eBWT)

1. define a new order relation (called ω-order) on the cyclic rotations [Mantaci, Restivo, R. and Sciortino, 2007];
2. append end-markers to each string and use the lexicographic order on the suffixes (called multi-string BWT or eBWT).

eBWT properties

- the strings belonging to S are not concatenated;
- reversible transformation (that produces a permutation of the symbols of the input string collection)
- produces a clustering effect (reduces the number of runs);
- strings can be added/removed (dynamic BWT);
- reconstruction of the entire collection or an its subset.

Multi-string BWT

We build the multi-string BWT:

- appending a distinct end-marker to each string of the collection S;
- without concatenating the strings in S;
- using the lexicographic order of the suffixes of the strings in the collection.

- Given strings collection $\mathsf{S} = \{S_1, S_2, \ldots, S_m\}$ on an alphabet Σ, one obtains the ordered collection:

$$\mathsf{S}' = \{S_1\$_1, S_2\$_2, \ldots, S_m\$_m\}$$

where

$$\$_1 < \$_2 < \cdots < \$_m < a, \text{ for each } a \in \Sigma \text{ and } \$_i \notin \Sigma \text{ for each } j = 1 \ldots m.$$

Remark

One can also obtain the BWT of a string collection in other ways "almost" equivalents.
Indeed, one could concatenate the input strings separating them with different end-markers and apply the single-string BWT.

How does multi-string BWT [Bauer et. al, CPM 2011, TCS 2013] work?

Given S $= \{GGAA, TCCT, GCCT, TTCT\}$:

- Sort all the suffixes (resp. cyclic rotations)[a] of the strings in $S' = \{S_i\$_i | S_i \in S\}$ (in our case: $S' = \{GGAA\$_1, TCCT\$_2, GCCT\$_3, TTCT\$_4\}$
- Output the string obtained by concatenating the symbols that (circularly) precede each first symbol of the suffixes (resp. last symbol of the rotations) in the sorted list.

Output:
$ebwt(S') = ATTTAGTGCCTG\$_3\$_1CCC\$_2T\$_4$.

Remark: Colors and Suffixes for clarity only.

[a] when appending a different dollar to the strings in S, the ω-order coincides with the lexicographical order.

Multi-string BWT	Sorted Suffixes	Sorted Cyclic Rotations
	$\$_1$	$\$_1GGAA$
	$\$_2$	$\$_2TCCT$
	$\$_3$	$\$_3GCCT$
	$\$_4$	$\$_4TTCT$
	$A\$_1$	$A\$_1GGA$
	$AA\$_1$	$AA\$_1GG$
	$CCT\$_2$	$CCT\$_2T$
	$CCT\$_3$	$CCT\$_3G$
	$CT\$_2$	$CT\$_2TC$
	$CT\$_3$	$CT\$_3GC$
	$CT\$_4$	$CT\$_4TT$
	$GAA\$_1$	$GAA\$_1G$
	$GCCT\$_3$	$GCCT\$_3$
	$GGAA\$_1$	$GGAA\$_1$
	$T\$_2$	$T\$_2TCC$
	$T\$_3$	$T\$_3GCC$
	$T\$_4$	$T\$_4TTC$
	$TCCT\$_2$	$TCCT\$_2$
	$TCT\$_4$	$TCT\$_4T$
	$TTCT\$_4$	$TTCT\$_4$

How does multi-string BWT [Bauer et. al, CPM 2011, TCS 2013] work?

Given $S = \{GGAA, TCCT, GCCT, TTCT\}$:

- Sort all the suffixes (resp. cyclic rotations)[a] of the strings in $S' = \{S_i\$_i | S_i \in S\}$ (in our case: $S' = \{GGAA\$_1, TCCT\$_2, GCCT\$_3, TTCT\$_4\}$
- Output the string obtained by concatenating the symbols that (circularly) precede each first symbol of the suffixes (resp. last symbol of the rotations) in the sorted list.

Output:
$ebwt(S') = ATTTAGTGCCTG\$_3\$_1CCC\$_2T\$_4.$

Remark: Colors and Suffixes for clarity only.

[a] when appending a different dollar to the strings in S, the ω-order coincides with the lexicographical order.

Multi-string BWT	Sorted Suffixes	Sorted Cyclic Rotations
	$\$_1$	$\$_1GGAA$
	$\$_2$	$\$_2TCCT$
	$\$_3$	$\$_3GCCT$
	$\$_4$	$\$_4TTCT$
	$A\$_1$	$A\$_1GGA$
	$AA\$_1$	$AA\$_1GG$
	$CCT\$_2$	$CCT\$_2T$
	$CCT\$_3$	$CCT\$_3G$
	$CT\$_2$	$CT\$_2TC$
	$CT\$_3$	$CT\$_3GC$
	$CT\$_4$	$CT\$_4TT$
	$GAA\$_1$	$GAA\$_1G$
	$GCCT\$_3$	$GCCT\$_3$
	$GGAA\$_1$	$GGAA\$_1$
	$T\$_2$	$T\$_2TCC$
	$T\$_3$	$T\$_3GCC$
	$T\$_4$	$T\$_4TTC$
	$TCCT\$_2$	$TCCT\$_2$
	$TCT\$_4$	$TCT\$_4T$
	$TTCT\$_4$	$TTCT\$_4$

How does multi-string BWT [Bauer et. al, CPM 2011, TCS 2013] work?

Given $S = \{GGAA, TCCT, GCCT, TTCT\}$:

- Sort all the suffixes (resp. cyclic rotations)[a] of the strings in $S' = \{S_i\$_i | S_i \in S\}$ (in our case: $S' = \{GGAA\$_1, TCCT\$_2, GCCT\$_3, TTCT\$_4\}$
- Output the string obtained by concatenating the symbols that (circularly) precede each first symbol of the suffixes (resp. last symbol of the rotations) in the sorted list.

Output:

$ebwt(S') = ATTTAGTGCCTG\$_3\$_1CCC\$_2T\$_4.$

Remark: Colors and Suffixes for clarity only.

[a] when appending a different dollar to the strings in S, the ω-order coincides with the lexicographical order.

Multi-string BWT	Sorted Suffixes	Sorted Cyclic Rotations
A	$\$_1$	$\$_1GGAA$
T	$\$_2$	$\$_2TCCT$
T	$\$_3$	$\$_3GCCT$
T	$\$_4$	$\$_4TTCT$
A	$A\$_1$	$A\$_1GGA$
G	$AA\$_1$	$AA\$_1GG$
T	$CCT\$_2$	$CCT\$_2T$
G	$CCT\$_3$	$CCT\$_3G$
C	$CT\$_2$	$CT\$_2TC$
C	$CT\$_3$	$CT\$_3GC$
T	$CT\$_4$	$CT\$_4TT$
G	$GAA\$_1$	$GAA\$_1G$
$\$_3$	$GCCT\$_3$	$GCCT\$_3$
$\$_1$	$GGAA\$_1$	$GGAA\$_1$
C	$T\$_2$	$T\$_2TCC$
C	$T\$_3$	$T\$_3GCC$
C	$T\$_4$	$T\$_4TTC$
$\$_2$	$TCCT\$_2$	$TCCT\$_2$
T	$TCT\$_4$	$TCT\$_4T$
$\$_4$	$TTCT\$_4$	$TTCT\$_4$

How does multi-string BWT [Bauer et. al, CPM 2011, TCS 2013] work?

Given $S = \{GGAA, TCCT, GCCT, TTCT\}$:

- Sort all the suffixes (resp. cyclic rotations)[a] of the strings in $S' = \{S_i\$_i | S_i \in S\}$ (in our case: $S' = \{GGAA\$_1, TCCT\$_2, GCCT\$_3, TTCT\$_4\}$
- Output the string obtained by concatenating the symbols that (circularly) precede each first symbol of the suffixes (resp. last symbol of the rotations) in the sorted list.

Output:
$ebwt(S') = ATTTAGTGCCTG\$_3\$_1CCC\$_2T\$_4.$

Remark: Colors and Suffixes for clarity only.

[a] when appending a different dollar to the strings in S, the ω-order coincides with the lexicographical order.

Multi-string BWT	Sorted Suffixes	Sorted Cyclic Rotations
A	$\$_1$	$\$_1GGAA$
T	$\$_2$	$\$_2TCCT$
T	$\$_3$	$\$_3GCCT$
T	$\$_4$	$\$_4TTCT$
A	$A\$_1$	$A\$_1GGA$
G	$AA\$_1$	$AA\$_1GG$
T	$CCT\$_2$	$CCT\$_2T$
G	$CCT\$_3$	$CCT\$_3G$
C	$CT\$_2$	$CT\$_2TC$
C	$CT\$_3$	$CT\$_3GC$
T	$CT\$_4$	$CT\$_4TT$
G	$GAA\$_1$	$GAA\$_1G$
$\$_3$	$GCCT\$_3$	$GCCT\$_3$
$\$_1$	$GGAA\$_1$	$GGAA\$_1$
C	$T\$_2$	$T\$_2TCC$
C	$T\$_3$	$T\$_3GCC$
C	$T\$_4$	$T\$_4TTC$
$\$_2$	$TCCT\$_2$	$TCCT\$_2$
T	$TCT\$_4$	$TCT\$_4T$
$\$_4$	$TTCT\$_4$	$TTCT\$_4$

Properties and Reversibility - LF mapping

- F is the concatenation of the first symbols of each suffix in the sorted list.
- The last symbol of S_j (just before the $\$_j$), for each $S_j \in$ S ($j = 1, \ldots, m$), is $L[j]$.
- LF Mapping: For each symbol t, the i-th occurrence of t in L corresponds to the i-th occurrence of t in F;

$$\pi_{LF} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 \\ 5 & 15 & 16 & 17 & 6 & 12 & 18 & 13 & 7 & 8 & 19 & 14 & 3 & 1 & 9 & 10 & 11 & 2 & 20 & 4 \end{pmatrix}$$

- For all $i = 1, \ldots, n$ the symbol $F[i]$ (circularly) follows $L[i]$ in the original (corresponding) string.

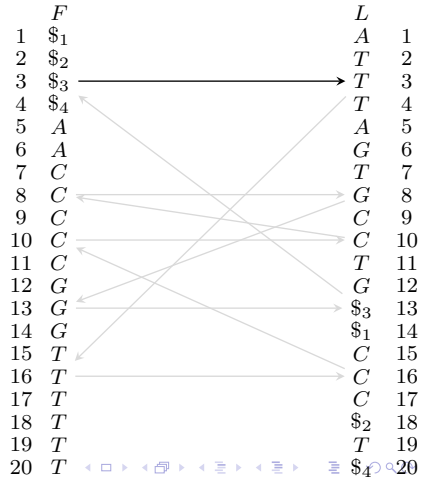
$S_3\$_3 =$ T

Cycle decomposition of π_{LF}:

$\pi_{LF} =$ (3)

$\{GGAA\$_1, TCCT\$_2, GCCT\$_3, TTCT\$_4\}$

$L = ATTTAGTGCCTG\$_3\$_1CCC\$_2T\$_4$



Properties and Reversibility - LF mapping

- F is the concatenation of the first symbols of each suffix in the sorted list.
- The last symbol of S_j (just before the $\$_j$), for each $S_j \in$ S ($j = 1, \ldots, m$), is $L[j]$.
- LF Mapping: For each symbol t, the i-th occurrence of t in L corresponds to the i-th occurrence of t in F;

$$\pi_{LF} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 \\ 5 & 15 & 16 & 17 & 6 & 12 & 18 & 13 & 7 & 8 & 19 & 14 & 3 & 1 & 9 & 10 & 11 & 2 & 20 & 4 \end{pmatrix}$$

- For all $i = 1, \ldots, n$ the symbol $F[i]$ (circularly) follows $L[i]$ in the original (corresponding) string.

$S_3\$_3 = \qquad T$

Cycle decomposition of π_{LF}:

$\pi_{LF} = \qquad (3\ 16 \qquad)$

$\{GGAA\$_1, TCCT\$_2, GCCT\$_3, TTCT\$_4\}$

$L = ATTTAGTGCCTG\$_3\$_1CCC\$_2T\$_4$

Properties and Reversibility - LF mapping

- F is the concatenation of the first symbols of each suffix in the sorted list.
- The last symbol of S_j (just before the $\$_j$), for each $S_j \in$ S ($j = 1, \ldots, m$), is $L[j]$.
- LF Mapping: For each symbol t, the i-th occurrence of t in L corresponds to the i-th occurrence of t in F;

$$\pi_{LF} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 \\ 5 & 15 & 16 & 17 & 6 & 12 & 18 & 13 & 7 & 8 & 19 & 14 & 3 & 1 & 9 & 10 & 11 & 2 & 20 & 4 \end{pmatrix}$$

- For all $i = 1, \ldots, n$ the symbol $F[i]$ (circularly) follows $L[i]$ in the original (corresponding) string.

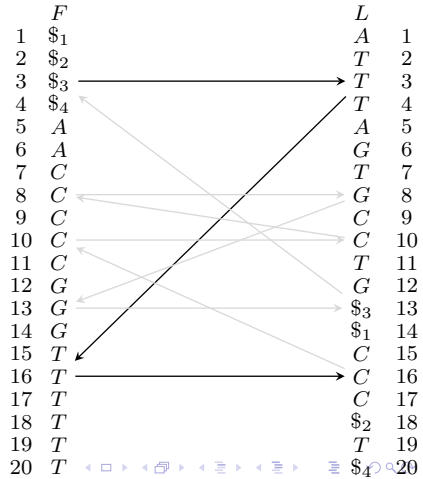
$S_3\$_3 = \quad CT$

Cycle decomposition of π_{LF}:

$\pi_{LF} = \quad (3\ 16 \quad)$

$\{GGAA\$_1, TCCT\$_2, GCCT\$_3, TTCT\$_4\}$

$L = ATTTAGTGCCTG\$_3\$_1CCC\$_2T\$_4$



Properties and Reversibility - LF mapping

- F is the concatenation of the first symbols of each suffix in the sorted list.
- The last symbol of S_j (just before the $\$_j$), for each $S_j \in$ S ($j = 1, \ldots, m$), is $L[j]$.
- LF Mapping: For each symbol t, the i-th occurrence of t in L corresponds to the i-th occurrence of t in F;

$$\pi_{LF} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 \\ 5 & 15 & 16 & 17 & 6 & 12 & 18 & 13 & 7 & 8 & 19 & 14 & 3 & 1 & 9 & 10 & 11 & 2 & 20 & 4 \end{pmatrix}$$

- For all $i = 1, \ldots, n$ the symbol $F[i]$ (circularly) follows $L[i]$ in the original (corresponding) string.

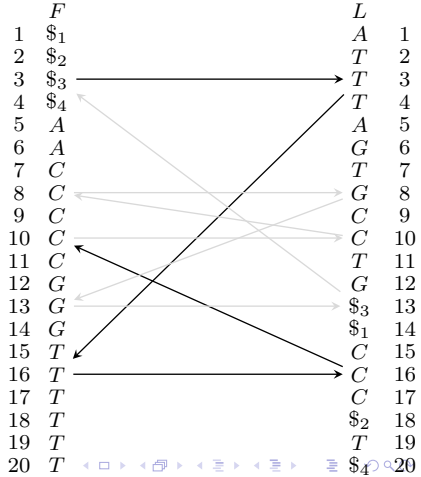
$S_3\$_3 = \quad CT$

Cycle decomposition of π_{LF}:

$\pi_{LF} = \quad (3\ 16\ 10 \quad)$

$\{GGAA\$_1, TCCT\$_2, GCCT\$_3, TTCT\$_4\}$

$L = ATTTAGTGCCTG\$_3\$_1CCC\$_2T\$_4$



Properties and Reversibility - LF mapping

- F is the concatenation of the first symbols of each suffix in the sorted list.
- The last symbol of S_j (just before the $\$_j$), for each $S_j \in$ S ($j = 1, \ldots, m$), is $L[j]$.
- LF Mapping: For each symbol t, the i-th occurrence of t in L corresponds to the i-th occurrence of t in F;

$$\pi_{LF} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 \\ 5 & 15 & 16 & 17 & 6 & 12 & 18 & 13 & 7 & 8 & 19 & 14 & 3 & 1 & 9 & 10 & 11 & 2 & 20 & 4 \end{pmatrix}$$

- For all $i = 1, \ldots, n$ the symbol $F[i]$ (circularly) follows $L[i]$ in the original (corresponding) string.

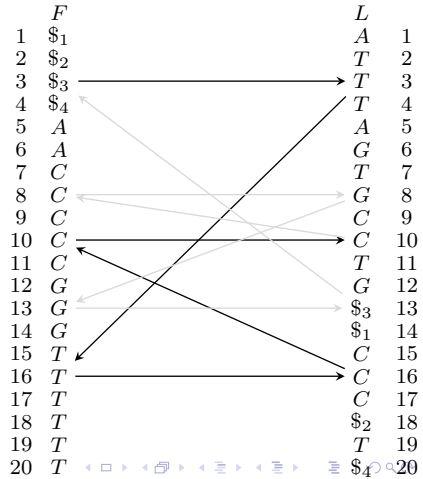
$S_3\$_3 = \quad CCT$

Cycle decomposition of π_{LF}:

$\pi_{LF} = \qquad (3\ 16\ 10 \qquad)$

$\{GGAA\$_1, TCCT\$_2, GCCT\$_3, TTCT\$_4\}$

$L = ATTTAGTGCCTG\$_3\$_1CCC\$_2T\$_4$



Properties and Reversibility - LF mapping

- F is the concatenation of the first symbols of each suffix in the sorted list.
- The last symbol of S_j (just before the $\$_j$), for each $S_j \in$ S ($j = 1, \ldots, m$), is $L[j]$.
- LF Mapping: For each symbol t, the i-th occurrence of t in L corresponds to the i-th occurrence of t in F;

$$\pi_{LF} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 \\ 5 & 15 & 16 & 17 & 6 & 12 & 18 & 13 & 7 & 8 & 19 & 14 & 3 & 1 & 9 & 10 & 11 & 2 & 20 & 4 \end{pmatrix}$$

- For all $i = 1, \ldots, n$ the symbol $F[i]$ (circularly) follows $L[i]$ in the original (corresponding) string.

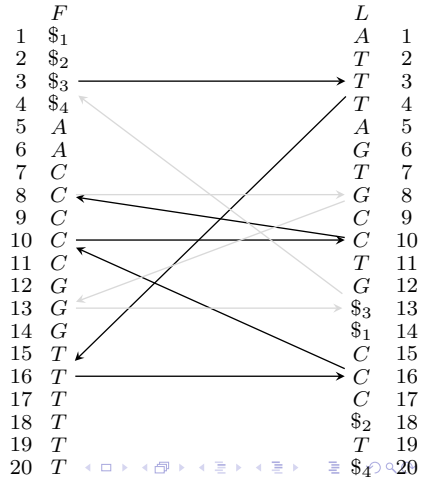
$S_3\$_3 = \quad CCT$

Cycle decomposition of π_{LF}:

$\pi_{LF} = \qquad (3\ 16\ 10\ 8 \quad)$

$\{GGAA\$_1, TCCT\$_2, GCCT\$_3, TTCT\$_4\}$

$L = ATTTAGTGCCTG\$_3\$_1CCC\$_2T\$_4$



Properties and Reversibility - LF mapping

- F is the concatenation of the first symbols of each suffix in the sorted list.
- The last symbol of S_j (just before the $\$_j$), for each $S_j \in$ S ($j = 1, \ldots, m$), is $L[j]$.
- LF Mapping: For each symbol t, the i-th occurrence of t in L corresponds to the i-th occurrence of t in F;

$$\pi_{LF} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 \\ 5 & 15 & 16 & 17 & 6 & 12 & 18 & 13 & 7 & 8 & 19 & 14 & 3 & 1 & 9 & 10 & 11 & 2 & 20 & 4 \end{pmatrix}$$

- For all $i = 1, \ldots, n$ the symbol $F[i]$ (circularly) follows $L[i]$ in the original (corresponding) string.

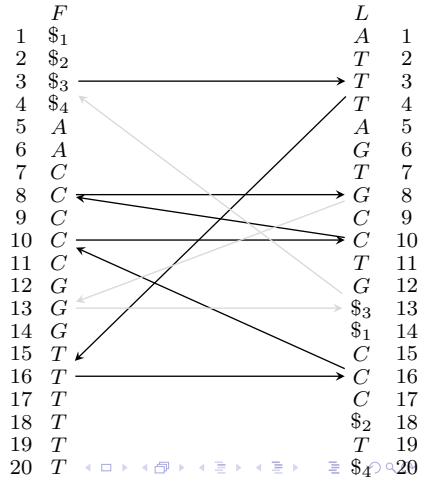
$S_3\$_3 = GCCT$

Cycle decomposition of π_{LF}:

$\pi_{LF} =$ (3 16 10 8)

$\{GGAA\$_1, TCCT\$_2, GCCT\$_3, TTCT\$_4\}$

$L = ATTTAGTGCCTG\$_3\$_1CCC\$_2T\$_4$



Properties and Reversibility - LF mapping

- F is the concatenation of the first symbols of each suffix in the sorted list.
- The last symbol of S_j (just before the $\$_j$), for each $S_j \in$ S ($j = 1, \ldots, m$), is $L[j]$.
- LF Mapping: For each symbol t, the i-th occurrence of t in L corresponds to the i-th occurrence of t in F;

$$\pi_{LF} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 \\ 5 & 15 & 16 & 17 & 6 & 12 & 18 & 13 & 7 & 8 & 19 & 14 & 3 & 1 & 9 & 10 & 11 & 2 & 20 & 4 \end{pmatrix}$$

- For all $i = 1, \ldots, n$ the symbol $F[i]$ (circularly) follows $L[i]$ in the original (corresponding) string.

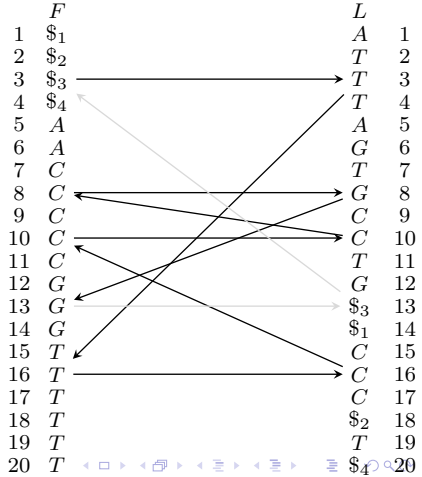
$S_3\$_3 = GCCT$

Cycle decomposition of π_{LF}:

$\pi_{LF} =$ (3 16 10 8 13)

$\{GGAA\$_1, TCCT\$_2, GCCT\$_3, TTCT\$_4\}$

$L = ATTTAGTGCCTG\$_3\$_1CCC\$_2T\$_4$



Properties and Reversibility - LF mapping

- F is the concatenation of the first symbols of each suffix in the sorted list.
- The last symbol of S_j (just before the $\$_j$), for each $S_j \in$ S ($j = 1, \ldots, m$), is $L[j]$.
- LF Mapping: For each symbol t, the i-th occurrence of t in L corresponds to the i-th occurrence of t in F;

$$\pi_{LF} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 \\ 5 & 15 & 16 & 17 & 6 & 12 & 18 & 13 & 7 & 8 & 19 & 14 & 3 & 1 & 9 & 10 & 11 & 2 & 20 & 4 \end{pmatrix}$$

- For all $i = 1, \ldots, n$ the symbol $F[i]$ (circularly) follows $L[i]$ in the original (corresponding) string.

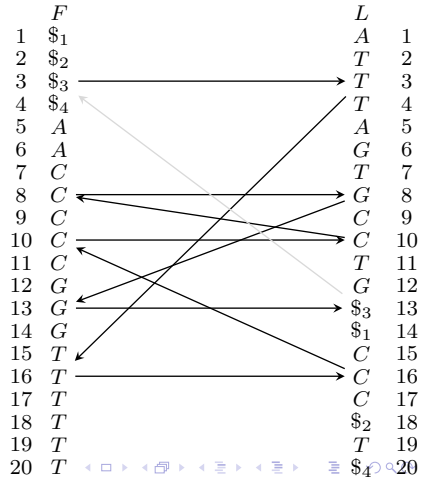
$S_3\$_3 = GCCT\$_3$

Cycle decomposition of π_{LF}:

$\pi_{LF} =$ (3 16 10 8 13)

$\{GGAA\$_1, TCCT\$_2, GCCT\$_3, TTCT\$_4\}$

$L = ATTTAGTGCCTG\$_3\$_1CCC\$_2T\$_4$



Properties and Reversibility - LF mapping

- F is the concatenation of the first symbols of each suffix in the sorted list.
- The last symbol of S_j (just before the $\$_j$), for each $S_j \in$ S ($j = 1, \ldots, m$), is $L[j]$.
- LF Mapping: For each symbol t, the i-th occurrence of t in L corresponds to the i-th occurrence of t in F;

$$\pi_{LF} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 \\ 5 & 15 & 16 & 17 & 6 & 12 & 18 & 13 & 7 & 8 & 19 & 14 & 3 & 1 & 9 & 10 & 11 & 2 & 20 & 4 \end{pmatrix}$$

- For all $i = 1, \ldots, n$ the symbol $F[i]$ (circularly) follows $L[i]$ in the original (corresponding) string.

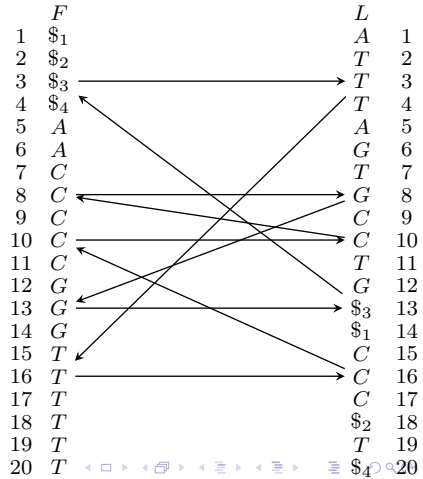
$S_3\$_3 = GCCT\$_3$

Cycle decomposition of π_{LF}:

$\pi_{LF} =$ (3 16 10 8 13)

$\{GGAA\$_1, TCCT\$_2, GCCT\$_3, TTCT\$_4\}$

$L = ATTTAGTGCCTG\$_3\$_1CCC\$_2T\$_4$



Properties and Reversibility - LF mapping

- F is the concatenation of the first symbols of each suffix in the sorted list.
- The last symbol of S_j (just before the $\$_j$), for each $S_j \in$ S ($j = 1, \ldots, m$), is $L[j]$.
- LF Mapping: For each symbol t, the i-th occurrence of t in L corresponds to the i-th occurrence of t in F;

$$\pi_{LF} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 \\ 5 & 15 & 16 & 17 & 6 & 12 & 18 & 13 & 7 & 8 & 19 & 14 & 3 & 1 & 9 & 10 & 11 & 2 & 20 & 4 \end{pmatrix}$$

- For all $i = 1, \ldots, n$ the symbol $F[i]$ (circularly) follows $L[i]$ in the original (corresponding) string.

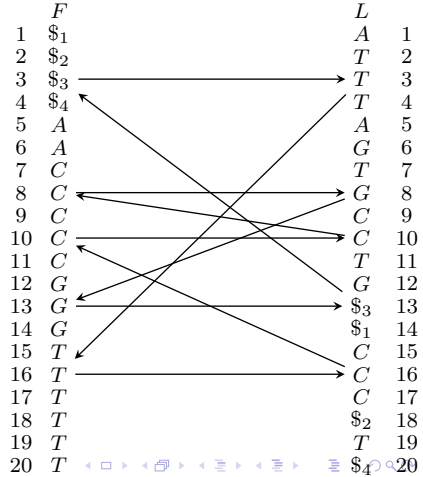
$S_3\$_3 = GCCT\$_3$

Cycle decomposition of π_{LF}:

$\pi_{LF} = (1\ 5\ 6\ 12\ 14)(2\ 15\ 9\ 7\ 18)(3\ 16\ 10\ 8\ 13)(4\ 17\ 11\ 19\ 20)$

$\{GGAA\$_1, TCCT\$_2, GCCT\$_3, TTCT\$_4\}$

$L = ATTTAGTGCCTG\$_3\$_1CCC\$_2T\$_4$



Compression of DNA string

First Goal

Compression of DNA bases by using multi-string BWT.

Why BWT and multi-string BWT?

Why?

- The motivation is the clustering effect that the BWT/eBWT produces, i.e. the BWT/eBWT reduces the number of the runs of the same symbol.
- The BWT/eBWT groups symbols with a similar context close together.

Example of clustering effect

When $v = mathematics$ (11 runs), we have:

$$bwt(v) = mmihttsecaa$$

Multi-string BWT

- we use m distinct end-markers for a collection of m strings;
- the collection is ordered.

Is it a problem in terms of the number of runs?

Why BWT and multi-string BWT?

Why?

- The motivation is the clustering effect that the BWT/eBWT produces, i.e. the BWT/eBWT reduces the number of the runs of the same symbol.
- The BWT/eBWT groups symbols with a similar context close together.

Example of clustering effect

When $v = mathematics$ (11 runs), we have:

$$bwt(v) = mmihttsecaa$$

Multi-string BWT

- we use m distinct end-markers for a collection of m strings;
- the collection is ordered.

Is it a problem in terms of the number of runs?

Why BWT and multi-string BWT?

Why?

- The motivation is the clustering effect that the BWT/eBWT produces, i.e. the BWT/eBWT reduces the number of the runs of the same symbol.
- The BWT/eBWT groups symbols with a similar context close together.

Example of clustering effect

When $v = mathematics$ (11 runs), we have:

$$bwt(v) = mmihttsecaa$$

Multi-string BWT

- we use m distinct end-markers for a collection of m strings;
- the collection is ordered.

Is it a problem in terms of the number of runs?

Why BWT and multi-string BWT?

Why?

- The motivation is the clustering effect that the BWT/eBWT produces, i.e. the BWT/eBWT reduces the number of the runs of the same symbol.
- The BWT/eBWT groups symbols with a similar context close together.

Example of clustering effect

When $v = mathematics$ (11 runs), we have:

$$bwt(v) = mmihttsecaa$$

Multi-string BWT

- we use m distinct end-markers for a collection of m strings;
- the collection is ordered.

Is it a problem in terms of the number of runs?

Why BWT and multi-string BWT?

Why?

- The motivation is the clustering effect that the BWT/eBWT produces, i.e. the BWT/eBWT reduces the number of the runs of the same symbol.
- The BWT/eBWT groups symbols with a similar context close together.

Example of clustering effect

When $v = mathematics$ (11 runs), we have:

$$bwt(v) = mmihttsecaa$$

Multi-string BWT

- we use m distinct end-markers for a collection of m strings;
- the collection is ordered.

Is it a problem in terms of the number of runs?

Why BWT and multi-string BWT?

Why?

- The motivation is the clustering effect that the BWT/eBWT produces, i.e. the BWT/eBWT reduces the number of the runs of the same symbol.
- The BWT/eBWT groups symbols with a similar context close together.

Example of clustering effect

When $v = mathematics$ (11 runs), we have:

$$bwt(v) = mmihttsecaa$$

Multi-string BWT

- we use m distinct end-markers for a collection of m strings;
- the collection is ordered.

Is it a problem in terms of the number of runs?

Why BWT and multi-string BWT?

Why?

- The motivation is the clustering effect that the BWT/eBWT produces, i.e. the BWT/eBWT reduces the number of the runs of the same symbol.
- The BWT/eBWT groups symbols with a similar context close together.

Example of clustering effect

When $v = mathematics$ (11 runs), we have:

$$bwt(v) = mmihttsecaa \text{ (8 runs)}$$

Multi-string BWT

- we use m distinct end-markers for a collection of m strings;
- the collection is ordered.

Is it a problem in terms of the number of runs?

Why BWT and multi-string BWT?

Why?

- The motivation is the clustering effect that the BWT/eBWT produces, i.e. the BWT/eBWT reduces the number of the runs of the same symbol.
- The BWT/eBWT groups symbols with a similar context close together.

Example of clustering effect

When $v = mathematics$ (11 runs), we have:

$$bwt(v) = mmihttsecaa \text{ (8 runs)}$$

Multi-string BWT

- we use m distinct end-markers for a collection of m strings;
- the collection is ordered.

Is it a problem in terms of the number of runs?

Distinct end-markers in multi-string BWT

First problem

The use of distinct end-marker symbols increases the size of the alphabet and makes compression more difficult.

Solution

We use implicit distinct end-markers, i.e. $\$_i = \$$ for each i: we use the position of the strings in the collection in order to establish the order relation between two identical suffixes:

$$\$_i < \$_j \text{ when } i < j.$$

Distinct end-markers in multi-string BWT

First problem

The use of distinct end-marker symbols increases the size of the alphabet and makes compression more difficult.

Solution

We use implicit distinct end-markers, i.e. $\$_i = \$$ for each i: we use the position of the strings in the collection in order to establish the order relation between two identical suffixes:

$$\$_i < \$_j \text{ when } i < j.$$

Ordered collection

Second problem

The use of ordered and (implicit or explicit) distinct end-marker symbols makes the multiset an ordered collection (the identical or similar sequences could be distant in the collection, by increasing the number of runs).

This can make the difference in the clustering effect (in terms of number of runs)!!!

Idea

We can reorder the strings reducing the number of runs!

eBWT	Sorted suffix
	...
T	GACA..
A	GACG..
A	**GATAG** $\$_p$
C	**GATAG** $\$_q$
A	**GATAG** $\$_r$
A	**GATAG** $\$_s$
C	**GATAG** $\$_t$
T	GATTTC..
T	GATTTGAT..
	...

where $p < q < r < s < t$

Ordered collection

Second problem

The use of ordered and (implicit or explicit) distinct end-marker symbols makes the multiset an ordered collection (the identical or similar sequences could be distant in the collection, by increasing the number of runs).

This can make the difference in the clustering effect (in terms of number of runs)!!!

Idea

We can reorder the strings reducing the number of runs!

eBWT	Sorted suffix
	...
T	GACA..
A	GACG..
A	**GATAG** $\$_p$
C	**GATAG** $\$_q$
A	**GATAG** $\$_r$
A	**GATAG** $\$_s$
C	**GATAG** $\$_t$
T	GATTTC..
T	GATTTGAT..
	...

where $p < q < r < s < t$

Example: Two different reordering of the input strings

$S = \{TAGA\underline{C}CT, TACC\underline{A}CT, GAGACCT\}$

$EBWT$	Sorted Suffixes
T	$
T	$
T	$
T	$ACCACT\$$
G	$ACCT\$$
G	$ACCT\$$
C	$ACT\$$
T	$AGACCT\$$
G	$AGACCT\$$
C	$CACT\$$
A	$CCACT\$$
A	$CCT\$$
A	$CCT\$$
$\underline{C}$	**CT**$
$\underline{A}$	**CT**$
C	**CT**$
A	$GACCT\$$
A	$GACCT\$$
$\$$	$GAGACCT\$$
C	$T\$$
C	$T\$$
C	$T\$$
$\$$	$TACCACT\$$
$\$$	$TAGACCT\$$

$S' = \{TACC\underline{A}CT, TAGA\underline{C}CT, GAGACCT\}$

$EBWT$	Sorted Suffixes
T	$
T	$
T	$
T	$ACCACT\$$
G	$ACCT\$$
G	$ACCT\$$
C	$ACT\$$
T	$AGACCT\$$
G	$AGACCT\$$
C	$CACT\$$
A	$CCACT\$$
A	$CCT\$$
A	$CCT\$$
$\underline{A}$	**CT**$
C	**CT**$
C	**CT**$
A	$GACCT\$$
A	$GACCT\$$
$\$$	$GAGACCT\$$
C	$T\$$
C	$T\$$
C	$T\$$
$\$$	$TACCACT\$$
$\$$	$TAGACCT\$$

Example: Two different reordering of the input strings

$\mathsf{S} = \{TAGA\underline{C}CT, TACC\underline{A}CT, GAGACCT\}$

$EBWT$	Sorted Suffixes
T	$
T	$
T	$
T	$ACCACT\$$
G	$ACCT\$$
G	$ACCT\$$
C	$ACT\$$
T	$AGACCT\$$
G	$AGACCT\$$
C	$CACT\$$
A	$CCACT\$$
A	$CCT\$$
A	$CCT\$$
$\underline{C}$	**CT$**
$\underline{A}$	**CT$**
C	**CT$**
A	$GACCT\$$
A	$GACCT\$$
$\$$	$GAGACCT\$$
C	$T\$$
C	$T\$$
C	$T\$$
$\$$	$TACCACT\$$
$\$$	$TAGACCT\$$

$\mathsf{S}' = \{TACC\underline{A}CT, TAGA\underline{C}CT, GAGACCT\}$

$EBWT$	Sorted Suffixes
T	$
T	$
T	$
T	$ACCACT\$$
G	$ACCT\$$
G	$ACCT\$$
C	$ACT\$$
T	$AGACCT\$$
G	$AGACCT\$$
C	$CACT\$$
A	$CCACT\$$
A	$CCT\$$
A	$CCT\$$
$\underline{A}$	**CT$**
$\underline{C}$	**CT$**
C	**CT$**
A	$GACCT\$$
A	$GACCT\$$
$\$$	$GAGACCT\$$
C	$T\$$
C	$T\$$
C	$T\$$
$\$$	$TACCACT\$$
$\$$	$TAGACCT\$$

Example: Two different reordering of the input strings

$S = \{TAGA\underline{C}CT, TACC\underline{A}CT, GAGACCT\}$

$EBWT$	Sorted Suffixes
T	$
T	$
T	$
T	$ACCACT\$$
G	$ACCT\$$
G	$ACCT\$$
C	$ACT\$$
T	$AGACCT\$$
G	$AGACCT\$$
C	$CACT\$$
A	$CCACT\$$
A	$CCT\$$
A	$CCT\$$
$\underline{C}$	**CT**$
$\underline{A}$	**CT**$
C	**CT**$
A	$GACCT\$$
A	$GACCT\$$
$	$GAGACCT\$$
C	$T\$$
C	$T\$$
C	$T\$$
$	$TACCACT\$$
$	$TAGACCT\$$

$S' = \{TACC\underline{A}CT, TAGA\underline{C}CT, GAGACCT\}$

$EBWT$	Sorted Suffixes
T	$
T	$
T	$
T	$ACCACT\$$
G	$ACCT\$$
G	$ACCT\$$
C	$ACT\$$
T	$AGACCT\$$
G	$AGACCT\$$
C	$CACT\$$
A	$CCACT\$$
A	$CCT\$$
A	$CCT\$$
$\underline{A}$	**CT**$
$\underline{C}$	**CT**$
C	**CT**$
A	$GACCT\$$
A	$GACCT\$$
$	$GAGACCT\$$
C	$T\$$
C	$T\$$
C	$T\$$
$	$TACCACT\$$
$	$TAGACCT\$$

SAP-interval and SAP array [Cox, Bauer, Jakobi and R., 2012]

Ordered collection: S = $\{TAGA\underline{C}CT, TACC\underline{A}CT, GAGACCT\}$

$EBWT$	$Suffixes$
T	$\$$
T	$\$$
T	$\$$
T	$ACCACT\$$
G	$ACCT\$$
G	$ACCT\$$
C	$ACT\$$
T	$AGACCT\$$
G	$AGACCT\$$
C	$CACT\$$
A	$CCACT\$$
A	$CCT\$$
A	$CCT\$$
C	$CT\$$
A	$CT\$$
C	$CT\$$
A	$GACCT\$$
A	$GACCT\$$
$\$$	$GAGACCT$
C	$T\$$
C	$T\$$
C	$T\$$
$\$$	$TACCACT$
$\$$	$TAGACCT\$$

Property

In regions of the eBWT, named SAP-interval (Same-As-Previous)[a] where the associated suffixes are the same, the ordering of the symbols in eBWT depends on the ordering of the strings in the collection.

[a] Related to SAP-intervals: the tuples described in [Bentley et al., ESA 2020] and the interesting intervals defined in [Cenzato and Lipták, CPM 2022].

The SAP-intervals can be represented as a binary array, called *SAP-array*: $SAP[i] = 1$ if $BWT[i]$ is associated with the suffix at position i (in the list of sorted suffixes) which is same as its previous suffix (at position $i-1$) up to the end-markers; and $SAP[i] = 0$ otherwise.

SAP-interval and SAP array [Cox, Bauer, Jakobi and R., 2012]

Ordered collection: $S = \{TAGA\underline{C}CT, TACC\underline{A}CT, GAGACCT\}$

$EBWT$	$Suffixes$
T	\$
T	\$
T	\$
T	$ACCACT\$$
G	$ACCT\$$
G	$ACCT\$$
C	$ACT\$$
T	$AGACCT\$$
G	$AGACCT\$$
C	$CACT\$$
A	$CCACT\$$
A	$CCT\$$
A	$CCT\$$
C	$CT\$$
A	$CT\$$
C	$CT\$$
A	$GACCT\$$
A	$GACCT\$$
\$	$GAGACCT$
C	$T\$$
C	$T\$$
C	$T\$$
\$	$TACCACT$
\$	$TAGACCT\$$

Property

In regions of the eBWT, named SAP-interval (Same-As-Previous)[a] where the associated suffixes are the same, the ordering of the symbols in eBWT depends on the ordering of the strings in the collection.

[a] Related to SAP-intervals: the tuples described in [Bentley et al., ESA 2020] and the interesting intervals defined in [Cenzato and Lipták, CPM 2022].

The SAP-intervals can be represented as a binary array, called *SAP-array*: $SAP[i] = 1$ if $BWT[i]$ is associated with the suffix at position i (in the list of sorted suffixes) which is same as its previous suffix (at position $i-1$) up to the end-markers; and $SAP[i] = 0$ otherwise.

SAP-interval and SAP array [Cox, Bauer, Jakobi and R., 2012]

Ordered collection: S = $\{TAGA\underline{C}CT, TACC\underline{A}CT, GAGACCT\}$

SAP-array	$EBWT$	$Suffixes$
0	T	\$
1	T	\$
1	T	\$
0	T	$ACCACT\$$
0	G	$ACCT\$$
1	G	$ACCT\$$
0	C	$ACT\$$
0	T	$AGACCT\$$
1	G	$AGACCT\$$
0	C	$CACT\$$
0	A	$CCACT\$$
0	A	$CCT\$$
1	A	$CCT\$$
0	C	$CT\$$
1	A	$CT\$$
1	C	$CT\$$
0	A	$GACCT\$$
1	A	$GACCT\$$
0	\$	$GAGACCT$
0	C	$T\$$
1	C	$T\$$
1	C	$T\$$
0	\$	$TACCACT$
1	\$	$TAGACCT\$$

Property

In regions of the eBWT, named SAP-interval (Same-As-Previous)[a] where the associated suffixes are the same, the ordering of the symbols in eBWT depends on the ordering of the strings in the collection.

[a] Related to SAP-intervals: the tuples described in [Bentley et al., ESA 2020] and the interesting intervals defined in [Cenzato and Lipták, CPM 2022].

The SAP-intervals can be represented as a binary array, called *SAP-array*: $SAP[i] = 1$ if $BWT[i]$ is associated with the suffix at position i (in the list of sorted suffixes) which is same as its previous suffix (at position $i-1$) up to the end-markers; and $SAP[i] = 0$ otherwise.

SAP-interval and SAP array [Cox, Bauer, Jakobi and R., 2012]

Ordered collection: $S = \{TACC\underline{A}CT, TAGA\underline{C}CT, GAGACCT\}$

SAP-array	$EBWT$	$Suffixes$
0	T	\$
1	T	\$
1	T	\$
0	T	$ACCACT\$$
0	G	$ACCT\$$
1	G	$ACCT\$$
0	C	$ACT\$$
0	T	$AGACCT\$$
1	G	$AGACCT\$$
0	C	$CACT\$$
0	A	$CCACT\$$
0	A	$CCT\$$
1	A	$CCT\$$
0	$\underline{A}$	$CT\$$
1	$\underline{C}$	$CT\$$
1	C	$CT\$$
0	A	$GACCT\$$
1	A	$GACCT\$$
0	\$	$GAGACCT$
0	C	$T\$$
1	C	$T\$$
1	C	$T\$$
0	\$	$TACCACT$
1	\$	$TAGACCT\$$

Property

In regions of the eBWT, named SAP-interval (Same-As-Previous)[a] where the associated suffixes are the same, the ordering of the symbols in eBWT depends on the ordering of the strings in the collection.

[a] Related to SAP-intervals: the tuples described in [Bentley et al., ESA 2020] and the interesting intervals defined in [Cenzato and Lipták, CPM 2022].

The SAP-intervals can be represented as a binary array, called *SAP-array*: $SAP[i] = 1$ if $BWT[i]$ is associated with the suffix at position i (in the list of sorted suffixes) which is same as its previous suffix (at position $i-1$) up to the end-markers; and $SAP[i] = 0$ otherwise.

How to reorder strings [Cox, Bauer, Jakobi and R., 2012]

$\{TAGACCT, TACCACT, GAGACCT\}$

$TACCACT, TAGACCT, GAGACCT\}$

SAP-array	$eBWT$	Sorted Suffixes
0	T	\$
1	T	\$
1	T	\$
0	T	$ACCACT\$$
0	G	$ACCT\$$
1	G	$ACCT\$$
0	C	$ACT\$$
0	T	$AGACCT\$$
1	G	$AGACCT\$$
0	C	$CACT\$$
0	A	$CCACT\$$
0	A	$CCT\$$
1	A	$CCT\$$
0	C A	$CT\$$
1	A C	$CT\$$
1	C	$CT\$$
0	A	$GACCT\$$
1	A	$GACCT\$$
0	\$	$GAGACCT\$$
0	C	$T\$$
1	C	$T\$$
1	C	$T\$$
0	\$	$TACCACT\$$
1	\$	$TAGACCT\$$

How can we reorder the strings reducing the number of runs?

- Pre-processing?
- No, reading both the BWT and its SAP-array, one can sort the symbols within the SAP interval and output a modified BWT.
- In alternative, one can reorder on-the-fly during the building of the eBWT. How?

By using BEETL-BCRext [Bauer, Cox and R., CPM 2011], we can swap the sequences $TAGACCT$ and $TACCACT$ in the ordered collection by swapping the symbols C and A directly in the eBWT during its construction [Cox, Bauer, Jakobi and R, 2012].

- The rest of eBWT is unaffected by this change in ordering
- lossless: the strings are not modified, we can only lose the original position of the strings in the collection.

How to reorder strings [Cox, Bauer, Jakobi and R., 2012]

$\{TAGACCT, TACCACT, GAGACCT\}$

$TACCACT, TAGACCT, GAGACCT\}$

SAP-array	$eBWT$	Sorted Suffixes
0	T	\$
1	T	\$
1	T	\$
0	T	$ACCACT$\$
0	G	$ACCT$\$
1	G	$ACCT$\$
0	C	ACT\$
0	T	$AGACCT$\$
1	G	$AGACCT$\$
0	C	$CACT$\$
0	A	$CCACT$\$
0	A	CCT\$
1	A	CCT\$
0	C A	CT\$
1	A C	CT\$
1	C	CT\$
0	A	$GACCT$\$
1	A	$GACCT$\$
0	\$	$GAGACCT$\$
0	C	T\$
1	C	T\$
1	C	T\$
0	\$	$TACCACT$\$
1	\$	$TAGACCT$\$

How can we reorder the strings reducing the number of runs?

- Pre-processing?
- No, reading both the BWT and its SAP-array, one can sort the symbols within the SAP interval and output a modified BWT.
- In alternative, one can reorder on-the-fly during the building of the eBWT. How?

By using BEETL-BCRext [Bauer, Cox and R., CPM 2011], we can swap the sequences $TAGACCT$ and $TACCACT$ in the ordered collection by swapping the symbols C and A directly in the eBWT during its construction [Cox, Bauer, Jakobi and R, 2012].

- The rest of eBWT is unaffected by this change in ordering
- lossless: the strings are not modified, we can only lose the original position of the strings in the collection.

How to reorder strings [Cox, Bauer, Jakobi and R., 2012]

$\{TAGACCT, TACCACT, GAGACCT\}$

$TACCACT, TAGACCT, GAGACCT\}$

SAP-array	$eBWT$	Sorted Suffixes
0	T	\$
1	T	\$
1	T	\$
0	T	$ACCACT\$$
0	G	$ACCT\$$
1	G	$ACCT\$$
0	C	$ACT\$$
0	T	$AGACCT\$$
1	G	$AGACCT\$$
0	C	$CACT\$$
0	A	$CCACT\$$
0	A	$CCT\$$
1	A	$CCT\$$
0	C A	$CT\$$
1	A C	$CT\$$
1	C	$CT\$$
0	A	$GACCT\$$
1	A	$GACCT\$$
0	\$	$GAGACCT\$$
0	C	$T\$$
1	C	$T\$$
1	C	$T\$$
0	\$	$TACCACT\$$
1	\$	$TAGACCT\$$

How can we reorder the strings reducing the number of runs?

- Pre-processing?
- No, reading both the BWT and its SAP-array, one can sort the symbols within the SAP interval and output a modified BWT.
- In alternative, one can reorder on-the-fly during the building of the eBWT. How?

By using BEETL-BCRext [Bauer, Cox and R., CPM 2011], we can swap the sequences $TAGACCT$ and $TACCACT$ in the ordered collection by swapping the symbols C and A directly in the eBWT during its construction [Cox, Bauer, Jakobi and R, 2012].

- The rest of eBWT is unaffected by this change in ordering
- lossless: the strings are not modified, we can only lose the original position of the strings in the collection.

How to reorder strings [Cox, Bauer, Jakobi and R., 2012]

$\{TAGACCT, TACCACT, GAGACCT\}$

$TACCACT, TAGACCT, GAGACCT\}$

SAP-array	$eBWT$	Sorted Suffixes
0	T	\$
1	T	\$
1	T	\$
0	T	$ACCACT\$$
0	G	$ACCT\$$
1	G	$ACCT\$$
0	C	$ACT\$$
0	T	$AGACCT\$$
1	G	$AGACCT\$$
0	C	$CACT\$$
0	A	$CCACT\$$
0	A	$CCT\$$
1	A	$CCT\$$
0	C A	$CT\$$
1	A C	$CT\$$
1	C	$CT\$$
0	A	$GACCT\$$
1	A	$GACCT\$$
0	\$	$GAGACCT\$$
0	C	$T\$$
1	C	$T\$$
1	C	$T\$$
0	\$	$TACCACT\$$
1	\$	$TAGACCT\$$

How can we reorder the strings reducing the number of runs?

- Pre-processing?
- No, reading both the BWT and its SAP-array, one can sort the symbols within the SAP interval and output a modified BWT.
- In alternative, one can reorder on-the-fly during the building of the eBWT. How?

By using BEETL-BCRext [Bauer, Cox and R., CPM 2011], we can swap the sequences $TAGACCT$ and $TACCACT$ in the ordered collection by swapping the symbols C and A directly in the eBWT during its construction [Cox, Bauer, Jakobi and R, 2012].

- The rest of eBWT is unaffected by this change in ordering
- lossless: the strings are not modified, we can only lose the original position of the strings in the collection.

How to reorder strings [Cox, Bauer, Jakobi and R., 2012]

$\{TAGACCT, TACCACT, GAGACCT\}$

$TACCACT, TAGACCT, GAGACCT\}$

SAP-array	$eBWT$	Sorted Suffixes
0	T	\$
1	T	\$
1	T	\$
0	T	$ACCACT\$$
0	G	$ACCT\$$
1	G	$ACCT\$$
0	C	$ACT\$$
0	T	$AGACCT\$$
1	G	$AGACCT\$$
0	C	$CACT\$$
0	A	$CCACT\$$
0	A	$CCT\$$
1	A	$CCT\$$
0	C A	$CT\$$
1	A C	$CT\$$
1	C	$CT\$$
0	A	$GACCT\$$
1	A	$GACCT\$$
0	\$	$GAGACCT\$$
0	C	$T\$$
1	C	$T\$$
1	C	$T\$$
0	\$	$TACCACT\$$
1	\$	$TAGACCT\$$

How can we reorder the strings reducing the number of runs?

- Pre-processing?
- No, reading both the BWT and its SAP-array, one can sort the symbols within the SAP interval and output a modified BWT.
- In alternative, one can reorder on-the-fly during the building of the eBWT. How?

By using BEETL-BCRext [Bauer, Cox and R., CPM 2011], we can swap the sequences $TAGACCT$ and $TACCACT$ in the ordered collection by swapping the symbols C and A directly in the eBWT during its construction [Cox, Bauer, Jakobi and R, 2012].

- The rest of eBWT is unaffected by this change in ordering
- lossless: the strings are not modified, we can only lose the original position of the strings in the collection.

How to reorder strings [Cox, Bauer, Jakobi and R., 2012]

$\{TAGA\underline{C}CT, TACC\underline{A}CT, GAGACCT\}$

$\downarrow$

$\{TACC\underline{A}CT, TAGA\underline{C}CT, GAGACCT\}$

SAP-array	$eBWT$	Sorted Suffixes
0	T	\$
1	T	\$
1	T	\$
0	T	$ACCACT$\$
0	G	$ACCT$\$
1	G	$ACCT$\$
0	C	ACT\$
0	T	$AGACCT$\$
1	G	$AGACCT$\$
0	C	$CACT$\$
0	A	$CCACT$\$
0	A	CCT\$
1	A	CCT\$
0	$\underline{C} \searrow \underline{A}$	CT\$
1	$\underline{A} \nearrow \underline{C}$	CT\$
1	C	CT\$
0	A	$GACCT$\$
1	A	$GACCT$\$
0	\$	$GAGACCT$\$
0	C	T\$
1	C	T\$
1	C	T\$
0	\$	$TACCACT$\$
1	\$	$TAGACCT$\$

How can we reorder the strings reducing the number of runs?

- Pre-processing?
- No, reading both the BWT and its SAP-array, one can sort the symbols within the SAP interval and output a modified BWT.
- In alternative, one can reorder on-the-fly during the building of the eBWT. How?

By using BEETL-BCRext [Bauer, Cox and R., CPM 2011], we can swap the sequences $TAGACCT$ and $TACCACT$ in the ordered collection by swapping the symbols C and A directly in the eBWT during its construction [Cox, Bauer, Jakobi and R, 2012].

- The rest of eBWT is unaffected by this change in ordering
- lossless: the strings are not modified, we can only lose the original position of the strings in the collection.

How to reorder strings [Cox, Bauer, Jakobi and R., 2012]

$\{TAGA\underline{C}CT, TACC\underline{A}CT, GAGACCT\}$

$\downarrow$

$\{TACC\underline{A}CT, TAGA\underline{C}CT, GAGACCT\}$

SAP-array	$eBWT$	Sorted Suffixes
0	T	\$
1	T	\$
1	T	\$
0	T	$ACCACT$\$
0	G	$ACCT$\$
1	G	$ACCT$\$
0	C	ACT\$
0	T	$AGACCT$\$
1	G	$AGACCT$\$
0	C	$CACT$\$
0	A	$CCACT$\$
0	A	CCT\$
1	A	CCT\$
0	$\underline{C} \searrow \underline{A}$	CT\$
1	$\underline{A} \nearrow \underline{C}$	CT\$
1	C	CT\$
0	A	$GACCT$\$
1	A	$GACCT$\$
0	\$	$GAGACCT$\$
0	C	T\$
1	C	T\$
1	C	T\$
0	\$	$TACCACT$\$
1	\$	$TAGACCT$\$

How can we reorder the strings reducing the number of runs?

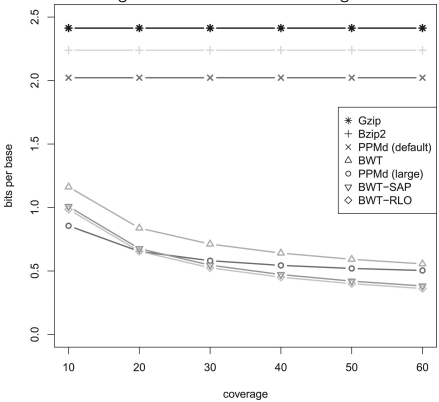
- Pre-processing?
- No, reading both the BWT and its SAP-array, one can sort the symbols within the SAP interval and output a modified BWT.
- In alternative, one can reorder on-the-fly during the building of the eBWT. How?

By using BEETL-BCRext [Bauer, Cox and R., CPM 2011], we can swap the sequences $TAGACCT$ and $TACCACT$ in the ordered collection by swapping the symbols C and A directly in the eBWT during its construction [Cox, Bauer, Jakobi and R, 2012].

- The rest of eBWT is unaffected by this change in ordering
- lossless: the strings are not modified, we can only lose the original position of the strings in the collection.

Experiments [Cox, Bauer, Jakobi and R., 2012]

$60\times$ coverage of error-free from the E.coli genome[a].



Gzip, Bzip2, PPMd (default) and PPMd (large) show compression achieved on the raw sequence data. BWT, BWT-SAP and BWT-RLO give compression results on the BWT using PPMd (default) as second-stage compressor.

[a] subsampled this into datasets as small as $10\times$

PPMd - $45\times$ human dataset[a]

	Input size	BWT	BWT-RLO	BWT-SAP
untrimmed	135.3 Gb	0.746	0.528	0.484
trimmed	133.6 Gb	0.721	0.504	0.462

Two heuristics that do not need to explicitly compute the SAP array, but modify EBWT construction algorithm by using an extra bit that tracks whether each suffix is "Same As Previous"):

Strategy RLO: (reverse lexicographic order, colex-order): This ensures EBWT symbols associated with such suffixes are grouped together (see [Heng Li, 2014] for an efficient implementation in internal memory, also for long reads).

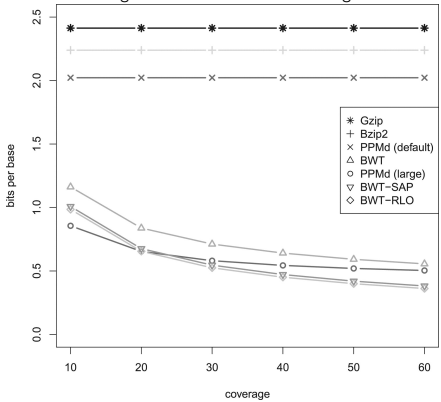
Strategy SAP: Approximation of the RLO: the symbols are not always permuted according to colex-order.

Outcome is EBWT of a permuted read collection.
Can verify by inverting the EBWT.

[a] Reads trimmed by following the strategy described for bwa which removed 1.3% of the bases.

Experiments [Cox, Bauer, Jakobi and R., 2012]

$60\times$ coverage of error-free from the E.coli genome[a].



Gzip, Bzip2, PPMd (default) and PPMd (large) show compression achieved on the raw sequence data. BWT, BWT-SAP and BWT-RLO give compression results on the BWT using PPMd (default) as second-stage compressor.

[a] subsampled this into datasets as small as $10\times$

PPMd - $45\times$ human dataset[a]

	Input size	BWT	BWT-RLO	BWT-SAP
untrimmed	135.3 Gb	0.746	0.528	0.484
trimmed	133.6 Gb	0.721	0.504	0.462

Two heuristics that do not need to explicitly compute the SAP array, but modify EBWT construction algorithm by using an extra bit that tracks whether each suffix is "Same As Previous"):

Strategy RLO: (reverse lexicographic order, colex-order): This ensures EBWT symbols associated with such suffixes are grouped together (see [Heng Li, 2014] for an efficient implementation in internal memory, also for long reads).

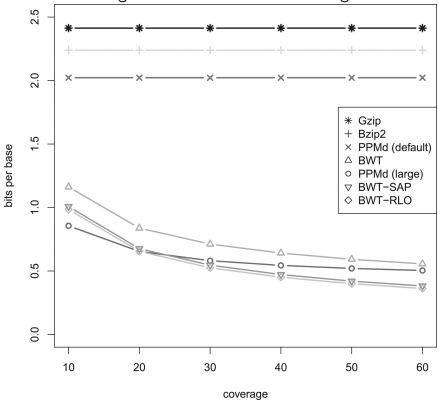
Strategy SAP: Approximation of the RLO: the symbols are not always permuted according to colex-order.

Outcome is EBWT of a permuted read collection.
Can verify by inverting the EBWT.

[a] Reads trimmed by following the strategy described for bwa which removed 1.3% of the bases.

Experiments [Cox, Bauer, Jakobi and R., 2012]

$60\times$ coverage of error-free from the E.coli genome[a].



Gzip, Bzip2, PPMd (default) and PPMd (large) show compression achieved on the raw sequence data. BWT, BWT-SAP and BWT-RLO give compression results on the BWT using PPMd (default) as second-stage compressor.

[a] subsampled this into datasets as small as $10\times$

PPMd - $45\times$ human dataset[a]

	Input size	BWT	BWT-RLO	BWT-SAP
untrimmed	135.3 Gb	0.746	0.528	0.484
trimmed	133.6 Gb	0.721	0.504	0.462

Two heuristics that do not need to explicitly compute the SAP array, but modify EBWT construction algorithm by using an extra bit that tracks whether each suffix is "Same As Previous"):

Strategy RLO: (reverse lexicographic order, colex-order): This ensures EBWT symbols associated with such suffixes are grouped together (see [Heng Li, 2014] for an efficient implementation in internal memory, also for long reads).

Strategy SAP: Approximation of the RLO: the symbols are not always permuted according to colex-order.

Outcome is EBWT of a permuted read collection.
Can verify by inverting the EBWT.

[a] Reads trimmed by following the strategy described for bwa which removed 1.3% of the bases.

Optimal BWT in terms of input order permutation

Can we swap the strings obtaining the minimum number of runs?

[Bentley, Gibney, and Thankachan, ESA 2020] show as compute the permutation of the input collection which yields the minimum number of runs of the resulting BWT.

One can compute the optimal BWT using the BWT and the SAP-array
(preliminary results in [Cenzato and Lipták, WCTA 2022])
Extended work: [Cenzato, Guerrini, Lipták and R., submitted].

Adaptive (lossy) compression of quality scores in BEETL [Janin, R. and Cox, 2014]

Second Goal

An adaptive and reference-free approach to lossy quality-score compression.

Insight

Discard the quality scores that are associated with bases that are "not interesting".

Insight

If a base in a read can, with high probability, be predicted by the context of bases that are next to it, then the base itself is imparting little additional information and its quality score can be discarded or aggressively compressed at little detriment to downstream analysis.

- **Q**: What do we mean by "not interesting"?
- **A**: How about "not likely to be important for downstream variant calling".

Adaptive (lossy) compression of quality scores in BEETL [Janin, R. and Cox, 2014]

Second Goal

An adaptive and reference-free approach to lossy quality-score compression.

Insight

Discard the quality scores that are associated with bases that are "not interesting".

Insight

If a base in a read can, with high probability, be predicted by the context of bases that are next to it, then the base itself is imparting little additional information and its quality score can be discarded or aggressively compressed at little detriment to downstream analysis.

- **Q**: What do we mean by "not interesting"?
- **A**: How about "not likely to be important for downstream variant calling".

Adaptive (lossy) compression of quality scores in BEETL [Janin, R. and Cox, 2014]

Second Goal

An adaptive and reference-free approach to lossy quality-score compression.

Insight

Discard the quality scores that are associated with bases that are "not interesting".

Insight

If a base in a read can, with high probability, be predicted by the context of bases that are next to it, then the base itself is imparting little additional information and its quality score can be discarded or aggressively compressed at little detriment to downstream analysis.

- **Q**: What do we mean by "not interesting"?
- **A**: How about "not likely to be important for downstream variant calling".

Adaptive (lossy) compression of quality scores in BEETL [Janin, R. and Cox, 2014]

Second Goal

An adaptive and reference-free approach to lossy quality-score compression.

Insight

Discard the quality scores that are associated with bases that are "not interesting".

Insight

If a base in a read can, with high probability, be predicted by the context of bases that are next to it, then the base itself is imparting little additional information and its quality score can be discarded or aggressively compressed at little detriment to downstream analysis.

- **Q**: What do we mean by "not interesting"?
- **A**: How about "not likely to be important for downstream variant calling".

Which scores to keep? [Janin, R. and Cox, 2014]

$Genoma$

$PEACHxBANANAxAPPLExPEARxTANGERINExORANGExPEACHxBANANAxPEAR$

Reads collection

$HxBANANAx$ $PLExPEARx$ $INExORANG$ $BANANAxPE$

$PEACHxBAN$ $PPLExPEAR$ $GERINExOR$ $HxBANANAx$

$BANANAxAP$ $PEARxTANG$ $RINExORAN$ $xPEACHxBA$

$EACHxBANA$ $LExPEARxT$ $ERINExORA$ $PEACHxBAN$

- $BANAN$ is always followed by A to make $BANANA$.
- Symbols that follow $BANAN$ are "not interesting".
- See $BANAN$ in a read $\rightarrow$ discard or smooth the quality score of next base.

- PEA could be the start of either $PEACH$ or $PEAR$.
- Symbols that follow PEA are "interesting".
- See PEA in a read $\rightarrow$ keep quality score of next base.

These patterns can be inferred from the reads, don't need to know genome.

Which scores to keep? [Janin, R. and Cox, 2014]

$Genoma$

$PEACHxBANANAxAPPLExPEARxTANGERINExORANGExPEACHxBANANAxPEAR$

Reads collection

$HxBANANAx$ $PLExPEARx$ $INExORANG$ $BANANAxPE$

$PEACHxBAN$ $PPLExPEAR$ $GERINExOR$ $HxBANANAx$

$BANANAxAP$ $PEARxTANG$ $RINExORAN$ $xPEACHxBA$

$EACHxBANA$ $LExPEARxT$ $ERINExORA$ $PEACHxBAN$

- $BANAN$ is always followed by A to make $BANANA$.
- Symbols that follow $BANAN$ are "not interesting".
- See $BANAN$ in a read $\rightarrow$ discard or smooth the quality score of next base.

- PEA could be the start of either $PEACH$ or $PEAR$.
- Symbols that follow PEA are "interesting".
- See PEA in a read $\rightarrow$ keep quality score of next base.

These patterns can be inferred from the reads, don't need to know genome.

Which scores to keep? [Janin, R. and Cox, 2014]

Genoma

$PEACHxBANANAxAPPLExPEARxTANGERINExORANGExPEACHxBANANAxPEAR$

Reads collection

$HxBANANAx$ $PLExPEARx$ $INExORANG$ $BANANAxPE$
$PEACHxBAN$ $PPLExPEAR$ $GERINExOR$ $HxBANANAx$
$BANANAxAP$ $PEARxTANG$ $RINExORAN$ $xPEACHxBA$
$EACHxBANA$ $LExPEARxT$ $ERINExORA$ $PEACHxBAN$

- $BANAN$ is always followed by A to make $BANANA$.
- Symbols that follow $BANAN$ are "not interesting".
- See $BANAN$ in a read $\rightarrow$ discard or smooth the quality score of next base.

- PEA could be the start of either $PEACH$ or $PEAR$.
- Symbols that follow PEA are "interesting".
- See PEA in a read $\rightarrow$ keep quality score of next base.

These patterns can be inferred from the reads, don't need to know genome.

Which scores to keep? [Janin, R. and Cox, 2014]

Genoma

$PEACHxBANANAxAPPLExPEARxTANGERINExORANGExPEACHxBANANAxPEAR$

Reads collection

$HxBANANAx$ $PLExPEARx$ $INExORANG$ $BANANAxPE$
$PEACHxBAN$ $PPLExPEAR$ $GERINExOR$ $HxBANANAx$
$BANANAxAP$ $PEARxTANG$ $RINExORAN$ $xPEACHxBA$
$EACHxBANA$ $LExPEARxT$ $ERINExORA$ $PEACHxBAN$

- $BANAN$ is always followed by A to make $BANANA$.
- Symbols that follow $BANAN$ are "not interesting".
- See $BANAN$ in a read → discard or smooth the quality score of next base.

- PEA could be the start of either $PEACH$ or $PEAR$.
- Symbols that follow PEA are "interesting".
- See PEA in a read → keep quality score of next base.

These patterns can be inferred from the reads, don't need to know genome.

Which scores to keep? [Janin, R. and Cox, 2014]

Genoma

$PEACHxBANANAxAPPLExPEARxTANGERINExORANGExPEACHxBANANAxPEAR$

Reads collection

$HxBANANAx$ $PLExPEARx$ $INExORANG$ $BANANAxPE$

$PEACHxBAN$ $PPLExPEAR$ $GERINExOR$ $HxBANANAx$

$BANANAxAP$ $PEARxTANG$ $RINExORAN$ $xPEACHxBA$

$EACHxBANA$ $LExPEARxT$ $ERINExORA$ $PEACHxBAN$

- $BANAN$ is always followed by A to make $BANANA$.
- Symbols that follow $BANAN$ are "not interesting".
- See $BANAN$ in a read $\rightarrow$ discard or smooth the quality score of next base.

- PEA could be the start of either $PEACH$ or $PEAR$.
- Symbols that follow PEA are "interesting".
- See PEA in a read $\rightarrow$ keep quality score of next base.

These patterns can be inferred from the reads, don't need to know genome.

Which scores to keep? [Janin, R. and Cox, 2014]

Genoma

$PEACHxBANANAxAPPLExPEARxTANGERINExORANGExPEACHxBANANAxPEAR$

Reads collection

$HxBANANAx$ $PLExPEARx$ $INExORANG$ $BANANAxPE$
$PEACHxBAN$ $PPLExPEAR$ $GERINExOR$ $HxBANANAx$
$BANANAxAP$ $PEARxTANG$ $RINExORAN$ $xPEACHxBA$
$EACHxBANA$ $LExPEARxT$ $ERINExORA$ $PEACHxBAN$

- $BANAN$ is always followed by A to make $BANANA$.
- Symbols that follow $BANAN$ are "not interesting".
- See $BANAN$ in a read $\rightarrow$ discard or smooth the quality score of next base.

- PEA could be the start of either $PEACH$ or $PEAR$.
- Symbols that follow PEA are "interesting".
- See PEA in a read $\rightarrow$ keep quality score of next base.

These patterns can be inferred from the reads, don't need to know genome.

Which scores to keep? [Janin, R. and Cox, 2014]

Genoma

$PEACHxBANANAxAPPLExPEARxTANGERINExORANGExPEACHxBANANAxPEAR$

Reads collection

$HxBANANAx$ $PLExPEARx$ $INExORANG$ $BANANAxPE$
$PEACHxBAN$ $PPLExPEAR$ $GERINExOR$ $HxBANANAx$
$BANANAxAP$ $PEARxTANG$ $RINExORAN$ $xPEACHxBA$
$EACHxBANA$ $LExPEARxT$ $ERINExORA$ $PEACHxBAN$

- $BANAN$ is always followed by A to make $BANANA$.
- Symbols that follow $BANAN$ are "not interesting".
- See $BANAN$ in a read → discard or smooth the quality score of next base.

- PEA could be the start of either $PEACH$ or $PEAR$.
- Symbols that follow PEA are "interesting".
- See PEA in a read → keep quality score of next base.

These patterns can be inferred from the reads, don't need to know genome.

Which scores to keep? [Janin, R. and Cox, 2014]

Genoma

$PEACHxBANANAxAPPLExPEARxTANGERINExORANGExPEACHxBANANAxPEAR$

Reads collection

$HxBANANAx$ $PLExPEARx$ $INExORANG$ $BANANAxPE$

$PEACHxBAN$ $PPLExPEAR$ $GERINExOR$ $HxBANANAx$

$BANANAxAP$ $PEARxTANG$ $RINExORAN$ $xPEACHxBA$

$EACHxBANA$ $LExPEARxT$ $ERINExORA$ $PEACHxBAN$

- $BANAN$ is always followed by A to make $BANANA$.
- Symbols that follow $BANAN$ are "not interesting".
- See $BANAN$ in a read $\rightarrow$ discard or smooth the quality score of next base.

- PEA could be the start of either $PEACH$ or $PEAR$.
- Symbols that follow PEA are "interesting".
- See PEA in a read $\rightarrow$ keep quality score of next base.

These patterns can be inferred from the reads, don't need to know genome.

Which scores to keep? [Janin, R. and Cox, 2014]

$$Genoma$$

$$PEACHxBANANAxAPPLExPEARxTANGERINExORANGExPEACHxBANANAxPEAR$$

Reads collection

$$HxBANANAx \quad PLExPEARx \quad INExORANG \quad BANANAxPE$$
$$PEACHxBAN \quad PPLExPEAR \quad GERINExOR \quad HxBANANAx$$
$$BANANAxAP \quad PEARxTANG \quad RINExORAN \quad xPEACHxBA$$
$$EACHxBANA \quad LExPEARxT \quad ERINExORA \quad PEACHxBAN$$

- $BANAN$ is always followed by A to make $BANANA$.
- Symbols that follow $BANAN$ are "not interesting".
- See $BANAN$ in a read $\rightarrow$ discard or smooth the quality score of next base.

- PEA could be the start of either $PEACH$ or $PEAR$.
- Symbols that follow PEA are "interesting".
- See PEA in a read $\rightarrow$ keep quality score of next base.

These patterns can be inferred from the reads, don't need to know genome.

QS string and LCP array

Let $S = \{S_1, S_2, \ldots, S_m\}$.

Let $\{S_1^q, S_2^q, \ldots, S_m^q\}$ be the ordered multi-set of associated quality scores.

LCP[i]: length of Longest Common Prefix between the i-th and the $(i-1)$-th suffix;

LCP-interval[i,j]: if $LCP[i] < c$, $LCP[h] \geq c$ for $h = i+1, \ldots, j$, $LCP[j+1] < c$.

QS[i]: quality score associated with eBWT[i];

QS(S')	LCP(S')	eBWT(S')	Sorted suffixes
=	0	G	$\$_1$
;	0	G	$\$_2$
¡	0	G	$\$_3$
?	0	T	ACATAG$\$_1$
!	4	T	ACATG$\$_3$
@	2	T	AG$\$_1$
	1	$\$_1$	ATACATAG$\$_1$
F	3	C	ATAG$\$_1$
+	2	C	ATG$\$_2$
¿	3	C	ATG$\$_3$
?	0	A	CATAG$\$_1$
	3	$\$_2$	CATG$\$_2$
@	3	A	CATG$\$_3$
B	0	A	G$\$_1$
;	1	T	G$\$_2$
F	1	T	G$\$_3$
,	0	A	TACATAG$\$_1$
	5	$\$_3$	TACATG$\$_3$
D	2	A	TAG$\$_1$
!	1	A	TG$\$_2$
&	1	A	TG$\$_3$

QS string and LCP array

Let $S = \{S_1, S_2, \ldots, S_m\}$.

Let $\{S_1^q, S_2^q, \ldots, S_m^q\}$ be the ordered multi-set of associated quality scores.

LCP[i]: length of Longest Common Prefix between the i-th and the $(i-1)$-th suffix;

LCP-interval[i,j]: if $LCP[i] < c$, $LCP[h] \geq c$ for $h = i+1, \ldots, j$, $LCP[j+1] < c$.

QS[i]: quality score associated with eBWT[i];

QS(S')	LCP(S')	eBWT(S')	Sorted suffixes
=	0	G	$\$_1$
;	0	G	$\$_2$
¡	0	G	$\$_3$
?	0	T	ACATAG$\$_1$
!	4	T	ACATG$\$_3$
@	2	T	AG$\$_1$
	1	$\$_1$	ATACATAG$\$_1$
F	3	C	ATAG$\$_1$
+	2	C	ATG$\$_2$
¿	3	C	ATG$\$_3$
?	0	A	CATAG$\$_1$
	3	$\$_2$	CATG$\$_2$
@	3	A	CATG$\$_3$
B	0	A	G$\$_1$
;	1	T	G$\$_2$
F	1	T	G$\$_3$
,	0	A	TACATAG$\$_1$
	5	$\$_3$	TACATG$\$_3$
D	2	A	TAG$\$_1$
!	1	A	TG$\$_2$
&	1	A	TG$\$_3$

QS string and LCP array

Let $S = \{S_1, S_2, \ldots, S_m\}$.

Let $\{S_1^q, S_2^q, \ldots, S_m^q\}$ be the ordered multi-set of associated quality scores.

LCP[i]: length of Longest Common Prefix between the i-th and the $(i-1)$-th suffix;

LCP-interval[i,j]: if $LCP[i] < c$, $LCP[h] \geq c$ for $h = i+1, \ldots, j$, $LCP[j+1] < c$.

QS[i]: quality score associated with eBWT[i];

QS(S')	LCP(S')	eBWT(S')	Sorted suffixes
=	0	G	$\$_1$
;	0	G	$\$_2$
¡	0	G	$\$_3$
?	0	T	ACATAG$\$_1$
!	4	T	ACATG$\$_3$
@	2	T	AG$\$_1$
	1	$\$_1$	ATACATAG$\$_1$
F	3	C	ATAG$\$_1$
+	2	C	ATG$\$_2$
¿	3	C	ATG$\$_3$
?	0	A	CATAG$\$_1$
	3	$\$_2$	CATG$\$_2$
@	3	A	CATG$\$_3$
B	0	A	G$\$_1$
;	1	T	G$\$_2$
F	1	T	G$\$_3$
,	0	A	TACATAG$\$_1$
	5	$\$_3$	TACATG$\$_3$
D	2	A	TAG$\$_1$
!	1	A	TG$\$_2$
&	1	A	TG$\$_3$

QS string and LCP array

Let $S = \{S_1, S_2, \ldots, S_m\}$.

Let $\{S_1^q, S_2^q, \ldots, S_m^q\}$ be the ordered multi-set of associated quality scores.

LCP[i]: length of Longest Common Prefix between the i-th and the $(i-1)$-th suffix;

LCP-interval[i,j]: if $LCP[i] < c$, $LCP[h] \geq c$ for $h = i+1, \ldots, j$, $LCP[j+1] < c$.

QS[i]: quality score associated with eBWT[i];

QS(S')	LCP(S')	eBWT(S')	Sorted suffixes
=	0	G	$\$_1$
;	0	G	$\$_2$
¡	0	G	$\$_3$
?	0	T	ACATAG$\$_1$
!	4	T	ACATG$\$_3$
@	2	T	AG$\$_1$
	1	$\$_1$	ATACATAG$\$_1$
F	3	C	ATAG$\$_1$
+	2	C	ATG$\$_2$
¿	3	C	ATG$\$_3$
?	0	A	CATAG$\$_1$
	3	$\$_2$	CATG$\$_2$
@	3	A	CATG$\$_3$
B	0	A	G$\$_1$
;	1	T	G$\$_2$
F	1	T	G$\$_3$
,	0	A	TACATAG$\$_1$
	5	$\$_3$	TACATG$\$_3$
D	2	A	TAG$\$_1$
!	1	A	TG$\$_2$
&	1	A	TG$\$_3$

Smoothing quality scores in BEETL [Janin, R. and Cox, 2014]

Sketch

Smoothing criteria based on parameters c, s:
IF LCP-value of LCP-interval $\geq c$
AND length of LCP-interval $\geq s$
AND all characters in LCP-interval are the same
THEN smooth

QS	eBWT	LCP	Sorted suffixes
	...		
;	T		GAC..
i	G	2	**GATA**CAT..
5	G	4	**GATA**GATA..
?	G	7	**GATA**GATTA..
=	G	8	**GATA**GATTT..
&	G	3	**GAT**TACAT..
@	G	5	**GATTA**GATA..
@	A	1	GCTTAGATA..
	...		

In this example $c = 3$ $s = 4$

Phrased in terms of the reads:

If any pattern of length c occurs at least s times and is always preceded by the same symbol, then smooth the quality scores of those occurrences of that symbol.

How to smooth?

We first compute the mean estimate error rate by converting each quality score to an error probability, taking the mean of these values and then converting back to Phred score (which we note is not the same as taking the mean of the quality scores).

Smoothing quality scores in BEETL [Janin, R. and Cox, 2014]

Sketch

Smoothing criteria based on parameters c, s:
IF LCP-value of LCP-interval $\geq c$
AND length of LCP-interval $\geq s$
AND all characters in LCP-interval are the same
THEN smooth

QS	eBWT	LCP	Sorted suffixes
	...		
;	T		GAC..
Q	G	2	**GATA**CAT..
Q	G	4	**GATA**GATA..
Q	G	7	**GATA**GATTA..
Q	G	8	**GATA**GATTT..
Q	G	3	**GAT**TACAT..
Q	G	5	**GATTA**GATA..
@	A	1	GCTTAGATA..
	...		

In this example $c = 3$ $s = 4$

Phrased in terms of the reads:

If any pattern of length c occurs at least s times and is always preceded by the same symbol, then smooth the quality scores of those occurrences of that symbol.

How to smooth?

We first compute the mean estimate error rate by converting each quality score to an error probability, taking the mean of these values and then converting back to Phred score (which we note is not the same as taking the mean of the quality scores).

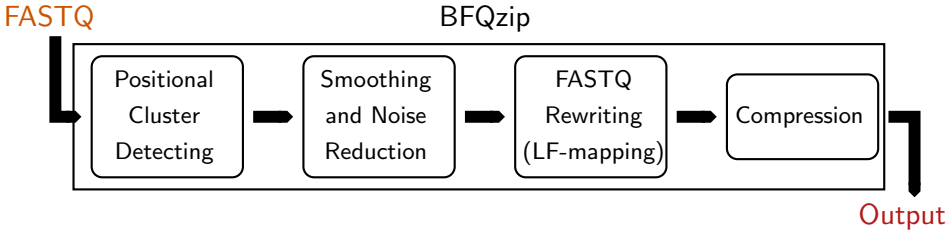
Smoothing QS with bases noise reduction [Guerrini, Louza and R., 2022]

Next goal

Compress a FASTQ file by

- smoothing the quality scores
- applying a noise reduction on corresponding bases,

while keeping variant calling performance comparable to original data.



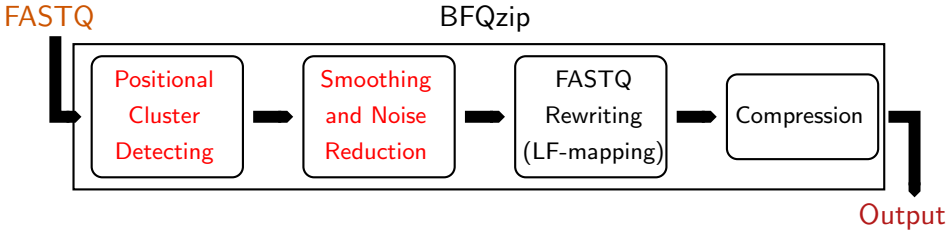
Smoothing QS with bases noise reduction [Guerrini, Louza and R., 2022]

Next goal

Compress a FASTQ file by

- smoothing the quality scores
- applying a noise reduction on corresponding bases,

while keeping variant calling performance comparable to original data.



Positional Clustering framework [Prezza, Pisanti, R. and Sciortino, 2019]

Designed to overcome the limitation of fixing a-priori the context length (for instance in the approaches based on LCP-interval).

A *eBWT positional cluster* eBWT$[i, j]$ is a maximal substring s.t for all $i < r \leq j$, $LCP[r]$ is not a *local minimum*.

- Automatically detects, in a **data-driven** way, the length k of the common context that differs cluster by cluster.
- Short random contexts can be excluded by setting a minimum value k_m.

Note. The value k_m and the shared context length k are likely to differ in most clusters.

eBWT	LCP	Sorted suffixes
G	0	$\$_1$
G	0	$\$_2$
G	0	$\$_3$
T	0	ACATAG$\$_1$
G	4	ACATG$\$_3$
T	1	AG$\$_1$
$\$_1$	2	AGACATAG$\$_1$
C	1	ATAG$\$_1$
C	2	ATG$\$_2$
C	3	ATG$\$_3$
A	0	CATAG$\$_1$
$\$_2$	3	CATG$\$_2$
A	4	CATG$\$_3$
A	0	G$\$_1$
T	1	G$\$_2$
T	1	G$\$_3$
A	1	GACATAG$\$_1$
$\$_3$	0	TACATG$\$_3$
A	2	TAG$\$_1$
A	1	TG$\$_2$
A	2	TG$\$_3$

Positional Clustering framework [Prezza, Pisanti, R. and Sciortino, 2019]

Designed to overcome the limitation of fixing a-priori the context length (for instance in the approaches based on LCP-interval).

A *eBWT positional cluster* eBWT$[i, j]$ is a maximal substring s.t for all $i < r \leq j$, $LCP[r]$ is not a *local minimum*.

- Automatically detects, in a **data-driven** way, the length k of the common context that differs cluster by cluster.
- Short random contexts can be excluded by setting a minimum value k_m.

Note. The value k_m and the shared context length k are likely to differ in most clusters.

eBWT	LCP	Sorted suffixes
G	0	$\$_1$
G	0	$\$_2$
G	0	$\$_3$
T	0	ACATAG$\$_1$
G	4	ACATG$\$_3$
T	1	AG$\$_1$
$\$_1$	2	AGACATAG$\$_1$
C	1	**AT**AG$\$_1$
C	2	**AT**G$\$_2$
C	3	**AT**G$\$_3$
A	0	CATAG$\$_1$
$\$_2$	3	CATG$\$_2$
A	4	CATG$\$_3$
A	0	G$\$_1$
T	1	G$\$_2$
T	1	G$\$_3$
A	1	GACATAG$\$_1$
$\$_3$	0	TACATG$\$_3$
A	2	TAG$\$_1$
A	1	TG$\$_2$
A	2	TG$\$_3$

Positional Clustering framework [Prezza, Pisanti, R. and Sciortino, 2019]

Designed to overcome the limitation of fixing a-priori the context length (for instance in the approaches based on LCP-interval).

A *eBWT positional cluster* eBWT$[i, j]$ is a maximal substring s.t for all $i < r \leq j$, $LCP[r]$ is not a *local minimum*.

- Automatically detects, in a **data-driven** way, the length k of the common context that differs cluster by cluster.
- Short random contexts can be excluded by setting a minimum value k_m.

Note. The value k_m and the shared context length k are likely to differ in most clusters.

eBWT	LCP	Sorted suffixes
G	0	$\$_1$
G	0	$\$_2$
G	0	$\$_3$
T	0	ACATAG$\$_1$
G	4	ACATG$\$_3$
T	1	AG$\$_1$
$\$_1$	2	AGACATAG$\$_1$
C	1	ATAG$\$_1$
C	2	ATG$\$_2$
C	3	ATG$\$_3$
A	0	CATAG$\$_1$
$\$_2$	3	CATG$\$_2$
A	4	CATG$\$_3$
A	0	G$\$_1$
T	1	G$\$_2$
T	1	G$\$_3$
A	1	GACATAG$\$_1$
$\$_3$	0	TACATG$\$_3$
A	2	TAG$\$_1$
A	1	TG$\$_2$
A	2	TG$\$_3$

Noise reduction

- We expect equal symbols inside positional clusters:

1. A *frequent symbol* is a symbol occurring in the cluster over some threshold.
2. A *noisy base* in a cluster C is a non-frequent symbol whose all occurrences in C have no high quality scores.

QS	eBWT	LCP	Sorted suffix
	...		
;	T		GACA..
F	A	3	GACG..
¡	C	2	**GATA**CAA..
E	C	4	**GATA**GATA..
?	C	7	**GATA**GATCA..
!	G	7	**GATA**GATTA..
=	C	5	**GATA**GG..
&	T	3	GATTACAT..
@	T	5	GATTAGATA..
	...		

Idea. Noisy bases are more likely noise introduced during sequencing.

$\Longrightarrow$ In any cluster, replace noisy bases with a predicted base.

Do not account for clusters with more than two frequent symbols.

Noise reduction

- We expect equal symbols inside positional clusters:

1. A *frequent symbol* is a symbol occurring in the cluster over some threshold.
2. A *noisy base* in a cluster C is a non-frequent symbol whose all occurrences in C have no high quality scores.

QS	eBWT	LCP	Sorted suffix
	...		
;	T		GACA..
F	A	3	GACG..
¡	C	2	**GATA**CAA..
E	C	4	**GATA**GATA..
?	C	7	**GATA**GATCA..
!	G	7	**GATA**GATTA..
=	C	5	**GATA**GG..
&	T	3	GATTACAT..
@	T	5	GATTAGATA..

Low quality score

Idea. Noisy bases are more likely noise introduced during sequencing.

$\Longrightarrow$ In any cluster, replace noisy bases with a predicted base.

Do not account for clusters with more than two frequent symbols.

Noise reduction: two cases

1. Unique frequent symbol ⇒ replace noisy bases with it.

QS	eBWT	LCP	Sorted suffix
	...		
;	T		GACA..
F	A	3	GACG..
¡	C	2	**GATA**CAA..
E	C	4	**GATA**GATA..
?	C	7	**GATA**GATCA..
!	G	7	**GATA**GATTA..
=	C	5	**GATA**GG..
&	T	3	GATTACAT..
@	T	5	GATTAGATA..
	...		

Noise reduction: two cases

1. Unique frequent symbol ⇒ replace noisy bases with it.

QS	eBWT	LCP	Sorted suffix
	...		
;	T		GACA..
F	A	3	GACG..
¡	C	2	**GATA**CAA..
E	C	4	**GATA**GATA..
?	C	7	**GATA**GATCA..
!	C	7	**GATA**GATTA..
=	C	5	**GATA**GG..
&	T	3	GATTACAT..
@	T	5	GATTAGATA..
	...		

Noise reduction: two cases

1. Unique frequent symbol ⇒ replace noisy bases with it.
2. Two different frequent symbols

QS	eBWT	LCP	Sorted suffixes
	...		
;	T		GACA..
F	A	3	GACG..
¿	A	2	**GATA**C..
¡	A	4	**GATA**G..
G	A	7	**GATA**GAC..
E	C	7	**GATA**GAGAA..
@	C	8	**GATA**GAGAT..
?	C	7	**GATA**GAGC..
!	G	7	**GATA**GAGTTA..
D	A	6	**GATA**GATTA
=	C	5	**GATA**GG..
&	T	3	GATTACAT..
@	T	5	GATTAG..
	...		

- Compute left contexts of considered bases (by LF-mapping).
- Replace any noisy base, if its left context coincides with all the left contexts of only one frequent symbol.

Noise reduction: two cases

1. Unique frequent symbol ⇒ replace noisy bases with it.
2. Two different frequent symbols

Left context	QS	eBWT	LCP	Sorted suffixes
		...		
	;	T		GACA..
	F	A	3	GACG..
CAT	¿	A	2	**GATA**C..
CAT	¡	A	4	**GATA**G..
CAT	G	A	7	**GATA**GAC..
ATA	E	C	7	**GATA**GAGAA..
ATA	@	C	8	**GATA**GAGAT..
ATA	?	C	7	**GATA**GAGC..
CAT	!	G	7	**GATA**GAGTTA..
CAT	D	A	6	**GATA**GATTA..
ATA	=	C	5	**GATA**GG..
	&	T	3	GATTACAT..
	@	T	5	GATTAG..
		...		

- Compute left contexts of considered bases (by LF-mapping).
- Replace any noisy base, if its left context coincides with all the left contexts of only one frequent symbol.

Noise reduction: two cases

1. Unique frequent symbol ⇒ replace noisy bases with it.
2. Two different frequent symbols

Left context	QS	eBWT	LCP	Sorted suffixes
		...		
	;	T		GACA..
	F	A	3	GACG..
CAT	¿	A	2	**GATA**C..
CAT	¡	A	4	**GATA**G..
CAT	G	A	7	**GATA**GAC..
ATA	E	C	7	**GATA**GAGAA..
ATA	@	C	8	**GATA**GAGAT..
ATA	?	C	7	**GATA**GAGC..
CAT	!	G	7	**GATA**GAGTTA..
CAT	D	A	6	**GATA**GATTA..
ATA	=	C	5	**GATA**GG..
	&	T	3	GATTACAT..
	@	T	5	GATTAG..
		...		

- Compute left contexts of considered bases (by LF-mapping).
- Replace any noisy base, if its left context coincides with all the left contexts of only one frequent symbol.

Noise reduction: two cases

1. Unique frequent symbol ⇒ replace noisy bases with it.
2. Two different frequent symbols

Left context	QS	eBWT	LCP	Sorted suffixes
		...		
	;	T		GACA..
	F	A	3	GACG..
CAT	¿	A	2	**GATA**C..
CAT	¡	A	4	**GATA**G..
CAT	G	A	7	**GATA**GAC..
ATA	E	C	7	**GATA**GAGAA..
ATA	@	C	8	**GATA**GAGAT..
ATA	?	C	7	**GATA**GAGC..
CAT	!	A	7	**GATA**GAGTTA..
CAT	D	A	6	**GATA**GATTA..
ATA	=	C	5	**GATA**GG..
	&	T	3	GATTACAT..
	@	T	5	GATTAG..
		...		

- Compute left contexts of considered bases (by LF-mapping).
- Replace any noisy base, if its left context coincides with all the left contexts of only one frequent symbol.

Smoothing quality score

- We expect quality scores inside positional clusters add little information
  $\Rightarrow$ smoothed over using a single value Q.

QS	eBWT	LCP	Sorted suffixes
	...		
;	T		CAT..
¡	G	0	**GATA**CAT..
5	G	4	**GATA**GATA..
?	G	7	**GATA**GATTA..
=	G	8	**GATA**GATTT..
&	T	3	GATTACAT..
@	A	5	GATTAGATA..
	...		

- The value Q can be computed according to four different strategies:
  - default value,
  - mean probability error,
  - maximum quality score,
  - average quality score.
- To reduce the number of the alphabet symbols, standard techniques (like Illumina 8-level binning) can be applied in addition to any above strategy.

Smoothing quality score

- We expect quality scores inside positional clusters add little information
  $\Rightarrow$ smoothed over using a single value Q.

QS	eBWT	LCP	Sorted suffixes
	...		
;	T		CAT..
Q	G	0	**GATA**CAT..
Q	G	4	**GATA**GATA..
Q	G	7	**GATA**GATTA..
Q	G	8	**GATA**GATTT..
&	T	3	GATTACAT..
@	A	5	GATTAGATA..
	...		

- The value Q can be computed according to four different strategies:
  - default value,
  - mean probability error,
  - maximum quality score,
  - average quality score.
- To reduce the number of the alphabet symbols, standard techniques (like Illumina 8-level binning) can be applied in addition to any above strategy.

Compression experiments - BFQzip tool [Guerrini, Louza and R., 2022]

- For comparison, two well-known compressors were used: PPMd and BSC.
- Paired-end datasets were compressed separately.

Compression ratio: $\frac{\text{compressed size}}{\text{original size}}$

Chr14 (18M reads, 101 length)		ERR262997_1 FASTQ	ERR262997_1 QS	ERR262997_1 DNA	ERR262997_2 FASTQ	ERR262997_2 QS	ERR262997_2 DNA
PPMd	Original	0.2482	0.2956	0.2100	0.2544	0.3076	0.2106
	LEON	0.1175	0.0301		0.1249	0.0444	
	BEETL	0.1916	0.1805		0.2010	0.1989	
	BFQzip	0.1957	0.1889	0.2098	0.2050	0.2074	0.2103
BSC	Original	0.1992	0.2862	0.1174	0.2071	0.2972	0.1224
	LEON	0.0674	0.0226		0.0770	0.0367	
	BEETL	0.1406	0.1698		0.1518	0.1874	
	BFQzip	0.1445	0.1786	0.1164	0.1555	0.1962	0.1210

- BEETL [Janin, R. and Cox, 2014] (based on eBWT, Reference-free and read-based),
- LEON [Benoit et. al, 2015] (assembly-based).
- All tested tools improved the compression of the original data.
- BFQzip and BEETL behaved similarly in all cases.
- LEON achieved a greater ability to smooth the quality scores, as it truncates all scores above a given threshold.

Validation - BFQZIP tool [Guerrini, Louza and R., 2022]

- Test the impact of modified data on single nucleotide polymorphisms (SNPs) discovery (BWA-MEM + HaplotypeCaller).
- Compare the set of called variants from each modified FASTQ with a baseline set:
  1. of variants obtained from the original FASTQ file;

	PREC (average %)	SEN (average %)	F (average %)
BEETL	96.020	95.360	95.690
LEON	96.027	93.617	94.802
BFQZIP	**96.303**	**95.373**	**95.837**

$$\mathrm{PREC} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}} \qquad \mathrm{SEN} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}} \qquad \mathrm{F} = \frac{2 \cdot \mathrm{SEN} \cdot \mathrm{PREC}}{\mathrm{SEN} + \mathrm{PREC}}$$

TP = variants matching in both baseline and called variants;
FP = variants in the called variants set but not in the baseline;
FN = variants missing in the called variants set but in the baseline.

BFQZIP reported a higher number of TP and the lowest number of FP.

Validation - BFQZIP tool [Guerrini, Louza and R., 2022]

- Test the impact of modified data on single nucleotide polymorphisms (SNPs) discovery (BWA-MEM + HaplotypeCaller).
- Compare the set of called variants from each modified FASTQ with a baseline set:
  1. of variants obtained from the original FASTQ file;

	PREC (average %)	SEN (average %)	F (average %)
BEETL	96.020	95.360	95.690
LEON	96.027	93.617	94.802
BFQZIP	**96.303**	**95.373**	**95.837**

$$\mathrm{PREC} = \frac{\mathrm{TP}}{\mathrm{TP+FP}} \qquad \mathrm{SEN} = \frac{\mathrm{TP}}{\mathrm{TP+FN}} \qquad \mathrm{F} = \frac{2 \cdot \mathrm{SEN} \cdot \mathrm{PREC}}{\mathrm{SEN} + \mathrm{PREC}}$$

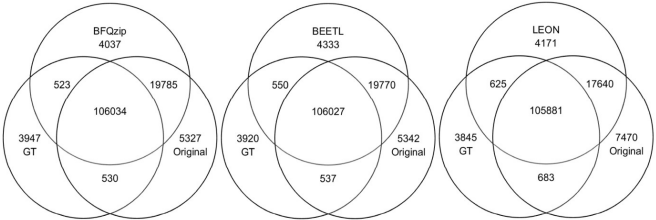
TP = variants matching in both baseline and called variants;
FP = variants in the called variants set but not in the baseline;
FN = variants missing in the called variants set but in the baseline.

BFQZIP reported a higher number of TP and the lowest number of FP.

Validation - BFQZIP tool [Guerrini, Louza and R., 2022]

- Test the impact of modified data on single nucleotide polymorphisms (SNPs) discovery (BWA-MEM + HaplotypeCaller).
- Compare the set of called variants from each modified FASTQ with a baseline set:
  1. of variants obtained from the original FASTQ file;
  2. "Ground Truth" for NA12878.

Ex. Chr14



BFQZIP preserved variants that are both in the original data and in the Ground Truth.

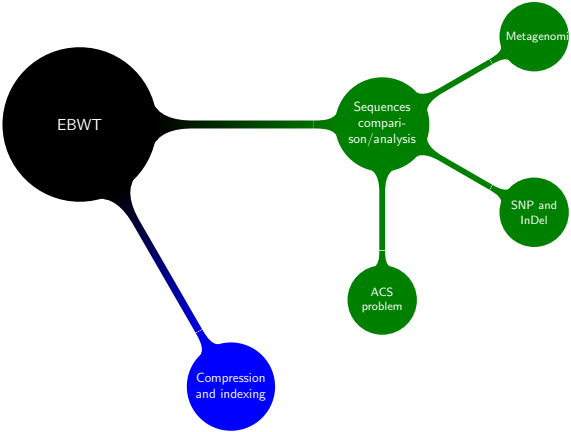
Further works

To introduce new eBWT-based compressors:

- Efficient Construction
- Indexing for other and newer comparison and analysis of sequences

Work in progress

Reordering reads. Combine the last approaches on FASTQ files with a reordering-based strategy, in a manner that "similar" reads are placed close together and can be encoded more efficiently.



For further reading I

Mohamed Ibrahim Abouelhoda, Stefan Kurtz, Enno Ohlebusch (2004).
Replacing suffix trees with enhanced suffix arrays.
Journal of Discrete Algorithms, 2(1):53–86.

Sabrina Mantaci, Antonio Restivo, G.R., and Marinella Sciortino (2007).
An extension of the Burrows-Wheeler Transform.
Theoret. Comput. Sci., 387(3):298–312.

Anthony J. Cox, Markus Bauer, and G.R. (2011).
Lightweight BWT construction for very large string collections.
In *CPM*, volume 6661 of *LNCS*, pages 219–231. Springer.

For further reading II

Anthony J. Cox, Markus Bauer, Tobias Jakobi, and G.R. (2012).
Large-scale compression of genomic sequence databases with the Burrows-Wheeler transform.
Bioinformatics, 28(11):1415–1419.

Markus Bauer, Anthony J. Cox, and G.R. (2013).
Lightweight algorithms for constructing and inverting the BWT of string collections.
Theoretical Computer Science, 483(0):134–148.

Lilian Janin, G.R., and Anthony J. Cox. (2014).
Adaptive reference-free compression of sequence quality scores.
Bioinformatics 30(1): 24–30,

For further reading III

Heng Li (2014).
Fast construction of FM-index for long sequence reads
Bioinformatics, 30(22):3274–3275.

G. Benoit, C. Lemaitre, D. Lavenier, E. Drezen, T. Dayris, R. Uricaru, G. Rizk. (2015).
Reference-free compression of high throughput sequencing data with a probabilistic de Bruijn graph.
BMC Bioinformatics, 2015, 16:288.

Anthony J. Cox, Fabio Garofalo, G.R., Marinella Sciortino, (2016).
Lightweight LCP construction for very large collections of strings.
In *Journal of Discrete Algorithms*, volume 37, pages 326–337. Springer.

For further reading IV

Nicola Prezza, Naida Pisanti, Marinella Sciortino, G.R. (2019)
SNPs detection by eBWT positional clustering.
Algorithms Mol Biol 14, 3.

Jason W. Bentley, Daniel Gibney, Sharma V. Thankachan (2020)
On the Complexity of BWT-Runs Minimization via Alphabet Reordering.
ESA, pages 15:1–15:13.

Davide Cenzato and Zsuzsanna Lipták (2022).
A Theoretical and Experimental Analysis of BWT Variants for String Collections
CPM, 25:1-25:18.

Veronica Guerrini, Felipe Louza and G.R. (2022).
Lossy Compressor Preserving Variant Calling through Extended BWT
BIOSTEC/BIOINFORMATICS, pages 38–48.

Most described algorithms are implemented in the Burrows-Wheeler Extended Tool Library (BEETL) library:

github.com:BEETL/BEETL.git

BFQzip:

github.com:veronicaguerrini/BFQzip.git

Thank you!