Extended Burrows-Wheeler Transform and analysis of biological sequences

Giovanna Rosone

Dipartimento di Matematica e Informatica
Università degli Studi di Palermo
Palermo, ITALY

Workshop on
“Combinatorial structures for sequence analysis in bionformatics”

Milano, 27th November 2013

Whole human genome sequencing

- Modern DNA sequencing machines produce a lot of data! e.g. Illumina HiSeq 2000: > 40Gbases of sequence per day (paired 100-mers).
- Datasets of 100 Gbases or more are common.

The Burrows-Wheeler Transform (BWT)

Many algorithms and data structures for compression and analysis of a sequence have the BWT at their heart.

- Traditionally the major application of the BWT has been for Data Compression.
- Today, there are reports of the application of the BWT in Bioinformatics, full-text compressed indexes, prediction and entropy estimation, and shape analysis in computer vision, etc.
- Many bioinformatics applications, e.g. the rapid search for maximal exact matches, shortest unique substrings and shortest absent words, use the Suffix Array (SA) and/or Burrows-Wheeler Transform (BWT) together with an additional table: the Longest Common Prefix (LCP) array.
- Together, SA/BWT and LCP can replace the larger suffix tree.

The Burrows-Wheeler Transform (BWT)

Example

- The BWT represents for instance the heart of the BZIP2 algorithm.
- BWT-based text indexes are the core of popular mapping programs
  1. Bowtie (Langmead et al.,Genome Biology 2009)
  2. BWA (Li and Durbin, Bioinformatics 2009, 2010)
  3. SOAP2 (Li et al., Bioinformatics 2009)
- *Simpson and Durbin*, Bioinformatics 2010: FM-index of a set of DNA sequences for overlap detection stage of de novo assembly;
- *Välimäki, Ladra and Mäkinen*, CPM 2010: Approximate All-Pairs Suffix/Prefix Overlaps.

The Burrows-Wheeler Transform (BWT)

Example

- The BWT represents for instance the heart of the BZIP2 algorithm.
- BWT-based text indexes are the core of popular mapping programs
  1. Bowtie (Langmead et al.,Genome Biology 2009)
  2. BWA (Li and Durbin, Bioinformatics 2009, 2010)
  3. SOAP2 (Li et al., Bioinformatics 2009)
- *Simpson and Durbin*, Bioinformatics 2010: FM-index of a set of DNA sequences for overlap detection stage of de novo assembly;
- *Välimäki, Ladra and Mäkinen*, CPM 2010: Approximate All-Pairs Suffix/Prefix Overlaps.

The Burrows-Wheeler Transform (BWT)

Example

- The BWT represents for instance the heart of the BZIP2 algorithm.
- BWT-based text indexes are the core of popular mapping programs
  1. Bowtie (Langmead et al.,Genome Biology 2009)
  2. BWA (Li and Durbin, Bioinformatics 2009, 2010)
  3. SOAP2 (Li et al., Bioinformatics 2009)
- *Simpson and Durbin*, Bioinformatics 2010: FM-index of a set of DNA sequences for overlap detection stage of de novo assembly;
- *Välimäki, Ladra and Mäkinen*, CPM 2010: Approximate All-Pairs Suffix/Prefix Overlaps.

The BWT

The BWT is a reversible transformation that produces a permutation of the letters in the input v (defined over an ordered alphabet Σ) so that occurrences of similar symbols tend to occur in clusters in the output sequence.

How does BWT work?

- BWT takes as input a text v, and produces:
  - a permutation $bwt(v)$ of the letters of v.
  - the index I, that is useful in order to recover the original word v.
- Example: $v = mathematics$

- Each row of the BW-matrix M is a conjugate of v in lexicographic order.
- $bwt(v)$ coincides with the last column L of M.
- The index I is the row of M containing the original sequence.
- Output: $bwt(v) = L = mmihttsecaa$ and $I = 7$.

$$M$$

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

Recall that two words $u, v \in \Sigma^*$ are conjugate, if $u = xy$ and $v = yx$ for some $x, y \in \Sigma^*$.

How does BWT work?

- BWT takes as input a text v, and produces:
  - a permutation $bwt(v)$ of the letters of v.
  - the index I, that is useful in order to recover the original word v.
- Example: $v = mathematics$

- Each row of the BW-matrix M is a conjugate of v in lexicographic order.
- $bwt(v)$ coincides with the last column L of M.
- The index I is the row of M containing the original sequence.
- Output: $bwt(v) = L = mmihttsecaa$ and $I = 7$.

$$M$$

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

Recall that two words $u, v \in \Sigma^*$, are conjugate, if $u = xy$ and $v = yx$ for some $x, y \in \Sigma^*$.

How does BWT work?

- BWT takes as input a text v, and produces:
  - a permutation $bwt(v)$ of the letters of v.
  - the index I, that is useful in order to recover the original word v.
- Example: $v = mathematics$

- Each row of the BW-matrix M is a conjugate of v in lexicographic order.
- $bwt(v)$ coincides with the last column L of M.
- The index I is the row of M containing the original sequence.
- Output: $bwt(v) = L = mmihttsecaa$ and $I = 7$.

$$M$$

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

Recall that two words $u, v \in \Sigma^*$ are conjugate, if $u = xy$ and $v = yx$ for some $x, y \in \Sigma^*$.

How does BWT work?

- BWT takes as input a text v, and produces:
  - a permutation $bwt(v)$ of the letters of v.
  - the index I, that is useful in order to recover the original word v.
- Example: $v = mathematics$

- Each row of the BW-matrix M is a conjugate of v in lexicographic order.
- $bwt(v)$ coincides with the last column L of M.
- The index I is the row of M containing the original sequence.
- Output: $bwt(v) = L = mmihttsecaa$ and $I = 7$.

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

Recall that two words $u, v \in \Sigma^*$ are conjugate, if $u = xy$ and $v = yx$ for some $x, y \in \Sigma^*$.

How does BWT work?

- BWT takes as input a text v, and produces:
  - a permutation $bwt(v)$ of the letters of v.
  - the index I, that is useful in order to recover the original word v.
- Example: $v = mathematics$

- Each row of the BW-matrix M is a conjugate of v in lexicographic order.
- $bwt(v)$ coincides with the last column L of M.
- The index I is the row of M containing the original sequence.
- Output: $bwt(v) = L = mmihttsecaa$ and $I = 7$.

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

Recall that two words $u, v \in \Sigma^*$ are conjugate, if $u = xy$ and $v = yx$ for some $x, y \in \Sigma^*$.

How does BWT work?

- BWT takes as input a text v, and produces:
  - a permutation $bwt(v)$ of the letters of v.
  - the index I, that is useful in order to recover the original word v.
- Example: $v = mathematics$

- Each row of the BW-matrix M is a conjugate of v in lexicographic order.
- $bwt(v)$ coincides with the last column L of M.
- The index I is the row of M containing the original sequence.
- Output: $bwt(v) = L = mmihttsecaa$ and $I = 7$.

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

Recall that two words $u, v \in \Sigma^*$ are conjugate, if $u = xy$ and $v = yx$ for some $x, y \in \Sigma^*$.

BWT and SA

When the symbol $\$$ is appended at the end of input string v (where $\$$ is unique and smaller than any other character), then one can sort the suffixes of $v\$$ rather than the conjugates of $v\$$.

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

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

M	L $\downarrow$	BWT	Sorted Suffixes
$\$mathematic$	s	s	$\$$
$athematics\$$	m	m	$athematics\$$
$atics\$mathe$	m	m	$atics\$$
$cs\$mathemat$	i	i	$cs\$$
$ematics\$mat$	h	h	$ematics\$$
$hematics\$ma$	t	t	$hematics\$$
$ics\$mathema$	t	t	$ics\$$
$mathematics$	$\$$	$\$$	$mathematics\$$
$matics\$math$	e	e	$matics\$$
$s\$mathemati$	c	c	$s\$$
$thematics\$m$	a	a	$thematics\$$
$tics\$mathem$	a	a	$tics\$$

Note that one can build the BWT of a string from its suffix array and vice versa.

BWT and SA

When the symbol $\$$ is appended at the end of input string v (where $\$$ is unique and smaller than any other character), then one can sort the suffixes of $v\$$ rather than the conjugates of $v\$$.

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

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

M	L $\downarrow$	SA	BWT	Sorted Suffixes
$\$\ m\ a\ t\ h\ e\ m\ a\ t\ i\ c$	s	12	s	$\$$
$a\ t\ h\ e\ m\ a\ t\ i\ c\ s\ \$$	m	2	m	$a\ t\ h\ e\ m\ a\ t\ i\ c\ s\ \$$
$a\ t\ i\ c\ s\ \$\ m\ a\ t\ h\ e$	m	7	m	$a\ t\ i\ c\ s\ \$$
$c\ s\ \$\ m\ a\ t\ h\ e\ m\ a\ t$	i	10	i	$c\ s\ \$$
$e\ m\ a\ t\ i\ c\ s\ \$\ m\ a\ t$	h	5	h	$e\ m\ a\ t\ i\ c\ s\ \$$
$h\ e\ m\ a\ t\ i\ c\ s\ \$\ m\ a$	t	4	t	$h\ e\ m\ a\ t\ i\ c\ s\ \$$
$i\ c\ s\ \$\ m\ a\ t\ h\ e\ m\ a$	t	9	t	$i\ c\ s\ \$$
$m\ a\ t\ h\ e\ m\ a\ t\ i\ c\ s$	$\$$	1	$\$$	$m\ a\ t\ h\ e\ m\ a\ t\ i\ c\ s\ \$$
$m\ a\ t\ i\ c\ s\ \$\ m\ a\ t\ h$	e	6	e	$m\ a\ t\ i\ c\ s\ \$$
$s\ \$\ m\ a\ t\ h\ e\ m\ a\ t\ i$	c	11	c	$s\ \$$
$t\ h\ e\ m\ a\ t\ i\ c\ s\ \$\ m$	a	3	a	$t\ h\ e\ m\ a\ t\ i\ c\ s\ \$$
$t\ i\ c\ s\ \$\ m\ a\ t\ h\ e\ m$	a	8	a	$t\ i\ c\ s\ \$$

Note that one can build the BWT of a string from its suffix array and viceversa.

Multiset of words

Next-generation DNA sequencing

The advent of “next-generation” DNA sequencing (NGS) technologies has meant that very large collections of DNA sequences are now commonplace in bioinformatics.

So, one could want to apply the algorithms based on the Burrows-Wheeler Transform to collections of sequences.

A classical method consists in:

- concatenating all strings of the collection separating each string with a distinct end-marker;
- computing the BWT of the string so obtained.

Multiset of words

Next-generation DNA sequencing

The advent of “next-generation” DNA sequencing (NGS) technologies has meant that very large collections of DNA sequences are now commonplace in bioinformatics.

So, one could want to apply the algorithms based on the Burrows-Wheeler Transform to collections of sequences.

A classical method consists in:

- concatenating all strings of the collection separating each string with a distinct end-marker;
- computing the BWT of the string so obtained.

Multiset of words

Next-generation DNA sequencing

The advent of "next-generation" DNA sequencing (NGS) technologies has meant that very large collections of DNA sequences are now commonplace in bioinformatics.

So, one could want to apply the algorithms based on the Burrows-Wheeler Transform to collections of sequences.

A classical method consists in:

- concatenating all strings of the collection separating each string with a distinct end-marker;
- computing the BWT of the string so obtained.

BWT of a collection of strings

One can ask whether it is possible to extend the notion of BWT to a multiset of words S:

- without concatenating the strings belonging to S,
- keeping the reversibility and the cluster effect,
- allowing sets of strings to be added/removed from collection,
- allowing the reconstruction of one or all sequences.

The answer is "yes". This problem has been faced in [Mantaci, Restivo, R. and Sciortino, 2005].
This transformation has been called "Extended Burrows-Wheeler Transform" (EBWT).

BWT of a collection of strings

One can ask whether it is possible to extend the notion of BWT to a multiset of words S:

- without concatenating the strings belonging to S,
- keeping the reversibility and the cluster effect,
- allowing sets of strings to be added/removed from collection,
- allowing the reconstruction of one or all sequences.

The answer is "yes". This problem has been faced in [Mantaci, Restivo, R. and Sciortino, 2005].
This transformation has been called "Extended Burrows-Wheeler Transform" (EBWT).

BWT of a collection of strings

One can ask whether it is possible to extend the notion of BWT to a multiset of words S:

- without concatenating the strings belonging to S,
- keeping the reversibility and the cluster effect,
- allowing sets of strings to be added/removed from collection,
- allowing the reconstruction of one or all sequences.

The answer is "yes". This problem has been faced in [Mantaci, Restivo, R. and Sciortino, 2005].
This transformation has been called "Extended Burrows-Wheeler Transform" (EBWT).

BWT of a collection of strings

One can ask whether it is possible to extend the notion of BWT to a multiset of words S:

- without concatenating the strings belonging to S,
- keeping the reversibility and the cluster effect,
- allowing sets of strings to be added/removed from collection,
- allowing the reconstruction of one or all sequences.

The answer is "yes". This problem has been faced in [Mantaci, Restivo, R. and Sciortino, 2005].
This transformation has been called "Extended Burrows-Wheeler Transform" (EBWT).

BWT of a collection of strings

One can ask whether it is possible to extend the notion of BWT to a multiset of words S:

- without concatenating the strings belonging to S,
- keeping the reversibility and the cluster effect,
- allowing sets of strings to be added/removed from collection,
- allowing the reconstruction of one or all sequences.

The answer is "yes". This problem has been faced in [Mantaci, Restivo, R. and Sciortino, 2005].
This transformation has been called "Extended Burrows-Wheeler Transform" (EBWT).

BWT of a collection of strings

One can ask whether it is possible to extend the notion of BWT to a multiset of words S:

- without concatenating the strings belonging to S,
- keeping the reversibility and the cluster effect,
- allowing sets of strings to be added/removed from collection,
- allowing the reconstruction of one or all sequences.

The answer is "yes". This problem has been faced in [Mantaci, Restivo, R. and Sciortino, 2005].
This transformation has been called "Extended Burrows-Wheeler Transform" (EBWT).

The Extended Burrows-Wheeler Transform [Mantaci, Restivo, R. and Sciortino, 2005]

- Sort all the conjugates of the words in S by the $\preceq_\omega$ order relation:

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

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

- Consider the list of the sorted conjugates 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.

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

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

Output:
$ebwt(S) = L = ccbbbcacaaabba$ and $\mathcal{I} = \{1, 9, 13, 14\}$.

The Extended Burrows-Wheeler Transform [Mantaci, Restivo, R. and Sciortino, 2005]

- Sort all the conjugates of the words in S by the $\preceq_\omega$ order relation:

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

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

- Consider the list of the sorted conjugates 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.

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

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

Output:
$ebwt(S) = L = ccbbbcacaaabba$ and $\mathcal{I} = \{1, 9, 13, 14\}$.

The Extended Burrows-Wheeler Transform [Mantaci, Restivo, R. and Sciortino, 2005]

- Sort all the conjugates of the words in S by the $\preceq_\omega$ order relation:

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

  where $u^\omega = uuuuu\cdots$ and $v^\omega = vvvvv\cdots$;
- Consider the list of the sorted conjugates 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.

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

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

Output:
$ebwt(\mathsf{S}) = L = ccbbbcacaaabba$ and $\mathcal{I} = \{1, 9, 13, 14\}$.

The Extended Burrows-Wheeler Transform [Mantaci, Restivo, R. and Sciortino, 2005]

- Sort all the conjugates of the words in S by the $\preceq_\omega$ order relation:

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

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

- Consider the list of the sorted conjugates 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.

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

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

Output:
$ebwt(\mathsf{S}) = L = ccbbbcacaaabba$ and
$\mathcal{I} = \{1, 9, 13, 14\}$.

The Extended Burrows-Wheeler Transform [Mantaci, Restivo, R. and Sciortino, 2005]

- Sort all the conjugates of the words in S by the $\preceq_\omega$ order relation:

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

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

- Consider the list of the sorted conjugates 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.

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

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

Output:
$ebwt(\mathsf{S}) = L = ccbbbcacaaabba$ and $\mathcal{I} = \{1, 9, 13, 14\}$.

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, R. and Sciortino, 2005], [Mantaci, Restivo, R. and Sciortino, 2007]: use a periodicity theorem to reduce the number of comparisons.
- [Hon, Ku, Lu, Shah and Thankachan, 2012]: a $O(n \log n)$ algorithm is provided, where n denotes the total length of the words in S.

EBWT for very large collection [Bauer, Cox and R., 2013]

Goal

Compute the EBWT of a collection of 1.000 millions of reads of length 100.

Solution

An efficient strategy (for computing the EBWT) by sorting the suffixes of all strings of the collection, by using the usual lexicographic order, has been given in [Bauer, Cox and R., 2011, Bauer, Cox and R., 2013], where:

the input collection and the output are in external memory!

- 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.
- Given strings collection S $= \{S_1, S_2, \dots, S_m\}$ on an alphabet Σ. We use (implicit distinct) end markers and suppose that

$$\$_1 < \$_2 < \cdots < \$_m < a, \text{ for each } a \in \Sigma.$$

EBWT for very large collection [Bauer, Cox and R., 2013]

Goal

Compute the EBWT of a collection of 1.000 millions of reads of length 100.

Solution

An efficient strategy (for computing the EBWT) by sorting the suffixes of all strings of the collection, by using the usual lexicographic order, has been given in [Bauer, Cox and R., 2011, Bauer, Cox and R., 2013], where:

the input collection and the output are in external memory!

- 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.
- Given strings collection S $= \{S_1, S_2, \ldots, S_m\}$ on an alphabet Σ. We use (implicit distinct) end markers and suppose that

$$\$_1 < \$_2 < \cdots < \$_m < a, \text{ for each } a \in \Sigma.$$

EBWT for very large collection [Bauer, Cox and R., 2013]

Goal

Compute the EBWT of a collection of 1.000 millions of reads of length 100.

Solution

An efficient strategy (for computing the EBWT) by sorting the suffixes of all strings of the collection, by using the usual lexicographic order, has been given in [Bauer, Cox and R., 2011, Bauer, Cox and R., 2013], where:

the input collection and the output are in external memory!

- 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.
- Given strings collection S $= \{S_1, S_2, \ldots, S_m\}$ on an alphabet Σ. We use (implicit distinct) end markers and suppose that

$$\$_1 < \$_2 < \cdots < \$_m < a, \text{ for each } a \in \Sigma.$$

Idea of the strategy by an example

Let S $= \{S_1, S_2, S_3\} = \{TGCCAAC, AGAGCTC, GTCGCTT\}$ be a collection of $m = 3$ strings of length $k = 7$ on an alphabet of $\sigma = 4$ letters.

	0	1	2	3	4	5	6	7
S_1								$\$_1$
S_2								$\$_2$
S_3								$\$_3$

We can obtain the EBWT of S by the following iterations:

Iteration 0	
$EBWT$	$Suffixes$
C	$\$_1$
C	$\$_2$
T	$\$_3$

Iteration 1	
$EBWT$	$Suffixes$
C	$\$_1$
C	$\$_2$
T	$\$_3$
A	$C\$_1$
T	$C\$_2$
T	$T\$_3$

Iteration 2	
$EBWT$	$Suffixes$
C	$\$_1$
C	$\$_2$
T	$\$_3$
A	$AC\$_1$
A	$C\$_1$
T	$C\$_2$
T	$T\$_3$
C	$TC\$_2$
C	$TT\$_3$

and so on

Idea of the strategy by an example

Let S $= \{S_1, S_2, S_3\} = \{TGCCAAC, AGAGCTC, GTCGCTT\}$ be a collection of $m = 3$ strings of length $k = 7$ on an alphabet of $\sigma = 4$ letters.

	0	1	2	3	4	5	6	7
S_1							C	$\$_1$
S_2							C	$\$_2$
S_3							T	$\$_3$

We can obtain the EBWT of S by the following iterations:

Iteration 0	
$EBWT$	$Suffixes$
C	$\$_1$
C	$\$_2$
T	$\$_3$

Iteration 1	
$EBWT$	$Suffixes$
C	$\$_1$
C	$\$_2$
T	$\$_3$
A	$C\$_1$
T	$C\$_2$
T	$T\$_3$

Iteration 2	
$EBWT$	$Suffixes$
C	$\$_1$
C	$\$_2$
T	$\$_3$
A	$AC\$_1$
A	$C\$_1$
T	$C\$_2$
T	$T\$_3$
C	$TC\$_2$
C	$TT\$_3$

and so on

Idea of the strategy by an example

Let S $= \{S_1, S_2, S_3\} = \{TGCCAAC, AGAGCTC, GTCGCTT\}$ be a collection of $m = 3$ strings of length $k = 7$ on an alphabet of $\sigma = 4$ letters.

	0	1	2	3	4	5	6	7
S_1						A	C	$\$_1$
S_2						T	C	$\$_2$
S_3						T	T	$\$_3$

We can obtain the EBWT of S by the following iterations:

Iteration 0	
$EBWT$	$Suffixes$
C	$\$_1$
C	$\$_2$
T	$\$_3$

Iteration 1	
$EBWT$	$Suffixes$
C	$\$_1$
C	$\$_2$
T	$\$_3$
A	$C\$_1$
T	$C\$_2$
T	$T\$_3$

Iteration 2	
$EBWT$	$Suffixes$
C	$\$_1$
C	$\$_2$
T	$\$_3$
A	$AC\$_1$
A	$C\$_1$
T	$C\$_2$
T	$T\$_3$
C	$TC\$_2$
C	$TT\$_3$

and so on

Idea of the strategy by an example

Let S $= \{S_1, S_2, S_3\} = \{TGCCAAC, AGAGCTC, GTCGCTT\}$ be a collection of $m = 3$ strings of length $k = 7$ on an alphabet of $\sigma = 4$ letters.

	0	1	2	3	4	5	6	7
S_1					A	A	C	$\$_1$
S_2					C	T	C	$\$_2$
S_3					C	T	T	$\$_3$

We can obtain the EBWT of S by the following iterations:

Iteration 0	
$EBWT$	$Suffixes$
C	$\$_1$
C	$\$_2$
T	$\$_3$

Iteration 1	
$EBWT$	$Suffixes$
C	$\$_1$
C	$\$_2$
T	$\$_3$
A	$C\$_1$
T	$C\$_2$
T	$T\$_3$

Iteration 2	
$EBWT$	$Suffixes$
C	$\$_1$
C	$\$_2$
T	$\$_3$
A	$AC\$_1$
A	$C\$_1$
T	$C\$_2$
T	$T\$_3$
C	$TC\$_2$
C	$TT\$_3$

and so on

Idea of the strategy by an example

Let S $= \{S_1, S_2, S_3\} = \{TGCCAAC, AGAGCTC, GTCGCTT\}$ be a collection of $m = 3$ strings of length $k = 7$ on an alphabet of $\sigma = 4$ letters.

	0	1	2	3	4	5	6	7
S_1				C	A	A	C	$\$_1$
S_2				G	C	T	C	$\$_2$
S_3				G	C	T	T	$\$_3$

We can obtain the EBWT of S by the following iterations:

Iteration 0	
$EBWT$	$Suffixes$
C	$\$_1$
C	$\$_2$
T	$\$_3$

Iteration 1	
$EBWT$	$Suffixes$
C	$\$_1$
C	$\$_2$
T	$\$_3$
A	$C\$_1$
T	$C\$_2$
T	$T\$_3$

Iteration 2	
$EBWT$	$Suffixes$
C	$\$_1$
C	$\$_2$
T	$\$_3$
A	$AC\$_1$
A	$C\$_1$
T	$C\$_2$
T	$T\$_3$
C	$TC\$_2$
C	$TT\$_3$

and so on

Idea of the strategy by an example

Let S $= \{S_1, S_2, S_3\} = \{TGCCAAC, AGAGCTC, GTCGCTT\}$ be a collection of $m = 3$ strings of length $k = 7$ on an alphabet of $\sigma = 4$ letters.

	0	1	2	3	4	5	6	7
S_1			C	C	A	A	C	$\$_1$
S_2			A	G	C	T	C	$\$_2$
S_3			C	G	C	T	T	$\$_3$

We can obtain the EBWT of S by the following iterations:

Iteration 0	
$EBWT$	$Suffixes$
C	$\$_1$
C	$\$_2$
T	$\$_3$

Iteration 1	
$EBWT$	$Suffixes$
C	$\$_1$
C	$\$_2$
T	$\$_3$
A	$C\$_1$
T	$C\$_2$
T	$T\$_3$

Iteration 2	
$EBWT$	$Suffixes$
C	$\$_1$
C	$\$_2$
T	$\$_3$
A	$AC\$_1$
A	$C\$_1$
T	$C\$_2$
T	$T\$_3$
C	$TC\$_2$
C	$TT\$_3$

and so on

Idea of the strategy by an example

Let S = $\{S_1, S_2, S_3\} = \{TGCCAAC, AGAGCTC, GTCGCTT\}$ be a collection of $m = 3$ strings of length $k = 7$ on an alphabet of $\sigma = 4$ letters.

	0	1	2	3	4	5	6	7
S_1		G	C	C	A	A	C	$\$_1$
S_2		G	A	G	C	T	C	$\$_2$
S_3		T	C	G	C	T	T	$\$_3$

We can obtain the EBWT of S by the following iterations:

Iteration 0	
$EBWT$	$Suffixes$
C	$\$_1$
C	$\$_2$
T	$\$_3$

Iteration 1	
$EBWT$	$Suffixes$
C	$\$_1$
C	$\$_2$
T	$\$_3$
A	$C\$_1$
T	$C\$_2$
T	$T\$_3$

Iteration 2	
$EBWT$	$Suffixes$
C	$\$_1$
C	$\$_2$
T	$\$_3$
A	$AC\$_1$
A	$C\$_1$
T	$C\$_2$
T	$T\$_3$
C	$TC\$_2$
C	$TT\$_3$

and so on

Idea of the strategy by an example

Let S $= \{S_1, S_2, S_3\} = \{TGCCAAC, AGAGCTC, GTCGCTT\}$ be a collection of $m = 3$ strings of length $k = 7$ on an alphabet of $\sigma = 4$ letters.

	0	1	2	3	4	5	6	7
S_1	T	G	C	C	A	A	C	$\$_1$
S_2	A	G	A	G	C	T	C	$\$_2$
S_3	G	T	C	G	C	T	T	$\$_3$

We can obtain the EBWT of S by the following iterations:

Iteration 0	
$EBWT$	$Suffixes$
C	$\$_1$
C	$\$_2$
T	$\$_3$

Iteration 1	
$EBWT$	$Suffixes$
C	$\$_1$
C	$\$_2$
T	$\$_3$
A	$C\$_1$
T	$C\$_2$
T	$T\$_3$

Iteration 2	
$EBWT$	$Suffixes$
C	$\$_1$
C	$\$_2$
T	$\$_3$
A	$AC\$_1$
A	$C\$_1$
T	$C\$_2$
T	$T\$_3$
C	$TC\$_2$
C	$TT\$_3$

and so on

Idea of the strategy by an example

Let S $= \{S_1, S_2, S_3\} = \{TGCCAAC, AGAGCTC, GTCGCTT\}$ be a collection of $m = 3$ strings of length $k = 7$ on an alphabet of $\sigma = 4$ letters.

	0	1	2	3	4	5	6	7
S_1	T	G	C	C	A	A	C	$\$_1$
S_2	A	G	A	G	C	T	C	$\$_2$
S_3	G	T	C	G	C	T	T	$\$_3$

We can obtain the EBWT of S by the following iterations:

Iteration 0

$EBWT$	$Suffixes$
C	$\$_1$
C	$\$_2$
T	$\$_3$

Iteration 1

$EBWT$	$Suffixes$
C	$\$_1$
C	$\$_2$
T	$\$_3$
A	$C\$_1$
T	$C\$_2$
T	$T\$_3$

Iteration 2

$EBWT$	$Suffixes$
C	$\$_1$
C	$\$_2$
T	$\$_3$
A	$AC\$_1$
A	$C\$_1$
T	$C\$_2$
T	$T\$_3$
C	$TC\$_2$
C	$TT\$_3$

and so on

Two versions of our algorithm: BCR vs. BCRext

Let S $= \{S_1, S_2, \ldots, S_m\}$ be a collection of strings of length k on an alphabet of σ letters.

	BCR	**BCRext**
CPU time	$O(k\text{sort}(m))$	$O(km)$
RAM usage(bits)	$O((m+\sigma^2)\log(mk))$	$O(\sigma^2 \log(mk))$
I/O (bits)	$O(mk^2 \log(s))$	$O(mk^2 \log(\sigma))$

Performance on human DNA sequence data.

Dataset size (millions of 100-mers)	*Program Program*	*Wallclock time (s per input base)*	*CPU efficiency (%)*	*Max RAM (Gbyte)*
85	bwte	7.99	99	4.00
	rlcsa	2.44	99	13.40
	BCR	1.01	83	1.10
	BCRext	4.75	27	negligible
1000	BCR	5.74	19	13.00
	BCRext	5.89	21	negligible

bwte: [*Ferragina, Gagie and Manzini*]'s algoritm ([Ferragina, Gagie and Manzini, 2012]).
rlcsa: [*Sirén*]'s algorithm ([Sirén, 2009]).

They does not support very large input collections.

Two versions of our algorithm: BCR vs. BCRext

Let S $= \{S_1, S_2, \ldots, S_m\}$ be a collection of strings of length k on an alphabet of σ letters.

	BCR	**BCRext**
CPU time	$O(k\mathrm{sort}(m))$	$O(km)$
RAM usage(bits)	$O((m+\sigma^2)\log(mk))$	$O(\sigma^2\log(mk))$
I/O (bits)	$O(mk^2\log(s))$	$O(mk^2\log(\sigma))$

Performance on human DNA sequence data.

Dataset size (millions of 100-mers)	*Program Program*	*Wallclock time (s per input base)*	*CPU efficiency (%)*	*Max RAM (Gbyte)*
85	bwte	7.99	99	4.00
	rlcsa	2.44	99	13.40
	BCR	1.01	83	1.10
	BCRext	4.75	27	negligible
1000	BCR	5.74	19	13.00
	BCRext	5.89	21	negligible

bwte: [*Ferragina, Gagie and Manzini*]'s algoritm ([Ferragina, Gagie and Manzini, 2012]).
rlcsa: [*Sirén*]'s algorithm ([Sirén, 2009]).

They does not support very large input collections.

EBWT, LCP and GSA for sequences collections [Bauer, Cox, R. and Sciortino, 2012]

Building upon the method (called `BCR`) of EBWT computation (in external memory) introduced in [Bauer, Cox and R., 2013], the algorithm in [Bauer, Cox, R. and Sciortino, 2012] adds some lightweight data structures and allows the LCP and EBWT of a collection of strings to be computed simultaneously.
Moreover, one can also compute the generalized suffix array at the same time.

Let S $= \{S_1, S_2, \ldots, S_m\}$ be a collection of strings on an alphabet of σ letters. The sum of lengths of S_i is N.

- $GSA[i]$: The i-th smallest suffix of the strings in S. If $GSA[i] = (t, h)$, then it corresponds to the suffix starting at the position t of the string S_h.
- $EBWT[i]$: The symbol that (circularly) precedes the first symbol of the suffix of S_h.
- $LCP[i]$: The length of longest common prefix with preceding suffix in the sorted list of the suffixes of S.

Example

Multiset S

	0	1	2	3	4	5	6
S_1	G	C	C	A	A	C	$\$_1$
S_2	G	A	G	C	T	C	$\$_2$
S_3	T	C	G	C	T	T	$\$_3$

	GSA	LCP	$EBWT$	Sorted Suffixes of S
0	$(6,1)$	0	C	$\$_1$
1	$(6,2)$	0	C	$\$_2$
2	$(6,3)$	0	T	$\$_3$
3	$(3,1)$	0	C	$AAC\$_1$
4	$(4,1)$	1	A	$AC\$_1$
5	$(1,2)$	1	G	$AGCTC\$_2$
6	$(5,1)$	0	A	$C\$_1$
7	$(5,2)$	1	T	$C\$_2$
8	$(2,1)$	1	C	$CAAC\$_1$
9	$(1,1)$	1	G	$CCAAC\$_1$
10	$(1,3)$	1	T	$CGCTT\$_3$
11	$(3,2)$	1	G	$CTC\$_2$
12	$(3,3)$	2	G	$CTT\$_3$
13	$(0,2)$	0	$\$_2$	$GAGCTC\$_2$
14	$(0,1)$	1	$\$_1$	$GCCAAC\$_1$
15	$(2,2)$	2	A	$GCTC\$_2$
16	$(2,3)$	3	C	$GCTT\$_3$
17	$(5,3)$	0	T	$T\$_3$
18	$(4,2)$	1	C	$TC\$_2$
19	$(0,3)$	2	$\$_3$	$TCGCTT\$_3$
20	$(4,3)$	1	C	$TT\$_3$

Let S $= \{S_1, S_2, \dots, S_m\}$ be a collection of strings on an alphabet of σ letters. The sum of lengths of S_i is N.

- $GSA[i]$: The i-th smallest suffix of the strings in S. If $GSA[i] = (t, h)$, then it corresponds to the suffix starting at the position t of the string S_h.
- $EBWT[i]$: The symbol that (circularly) precedes the first symbol of the suffix of S_h.
- $LCP[i]$: The length of longest common prefix with preceding suffix in the sorted list of the suffixes of S.

Example

Multiset S

	0	1	2	3	4	5	6
S_1	G	C	C	A	A	C	$\$_1$
S_2	G	A	G	C	T	C	$\$_2$
S_3	T	C	G	C	T	T	$\$_3$

	GSA	LCP	$EBWT$	Sorted Suffixes of S
0	(6, 1)	0	C	$\$_1$
1	(6, 2)	0	C	$\$_2$
2	(6, 3)	0	T	$\$_3$
3	(3, 1)	0	C	$AAC\$_1$
4	(4, 1)	1	A	$AC\$_1$
5	(1, 2)	1	G	$AGCTC\$_2$
6	(5, 1)	0	A	$C\$_1$
7	(5, 2)	1	T	$C\$_2$
8	(2, 1)	1	C	$CAAC\$_1$
9	(1, 1)	1	G	$CCAAC\$_1$
10	(1, 3)	1	T	$CGCTT\$_3$
11	(3, 2)	1	G	$CTC\$_2$
12	(3, 3)	2	G	$CTT\$_3$
13	(0, 2)	0	$\$_2$	$GAGCTC\$_2$
14	(0, 1)	1	$\$_1$	$GCCAAC\$_1$
15	(2, 2)	2	A	$GCTC\$_2$
16	(2, 3)	3	C	$GCTT\$_3$
17	(5, 3)	0	T	$T\$_3$
18	(4, 2)	1	C	$TC\$_2$
19	(0, 3)	2	$\$_3$	$TCGCTT\$_3$
20	(4, 3)	1	C	$TT\$_3$

Let S $= \{S_1, S_2, \ldots, S_m\}$ be a collection of strings on an alphabet of σ letters. The sum of lengths of S_i is N.

- $GSA[i]$: The i-th smallest suffix of the strings in S. If $GSA[i] = (t, h)$, then it corresponds to the suffix starting at the position t of the string S_h.
- $EBWT[i]$: The symbol that (circularly) precedes the first symbol of the suffix of S_h.
- $LCP[i]$: The length of longest common prefix with preceding suffix in the sorted list of the suffixes of S.

Example

Multiset S

	0	1	2	3	4	5	6
S_1	G	C	C	A	A	C	$\$_1$
S_2	G	A	G	C	T	C	$\$_2$
S_3	T	C	G	C	T	T	$\$_3$

	GSA	LCP	$EBWT$	Sorted Suffixes of S
0	$(6,1)$	0	C	$\$_1$
1	$(6,2)$	0	C	$\$_2$
2	$(6,3)$	0	T	$\$_3$
3	$(3,1)$	0	C	$AAC\$_1$
4	$(4,1)$	1	A	$AC\$_1$
5	$(1,2)$	1	G	$AGCTC\$_2$
6	$(5,1)$	0	A	$C\$_1$
7	$(5,2)$	1	T	$C\$_2$
8	$(2,1)$	1	C	$CAAC\$_1$
9	$(1,1)$	1	G	$CCAAC\$_1$
10	$(1,3)$	1	T	$CGCTT\$_3$
11	$(3,2)$	1	G	$CTC\$_2$
12	$(3,3)$	2	G	$CTT\$_3$
13	$(0,2)$	0	$\$_2$	$GAGCTC\$_2$
14	$(0,1)$	1	$\$_1$	$GCCAAC\$_1$
15	$(2,2)$	2	A	$GCTC\$_2$
16	$(2,3)$	3	C	$GCTT\$_3$
17	$(5,3)$	0	T	$T\$_3$
18	$(4,2)$	1	C	$TC\$_2$
19	$(0,3)$	2	$\$_3$	$TCGCTT\$_3$
20	$(4,3)$	1	C	$TT\$_3$

Let S $= \{S_1, S_2, \ldots, S_m\}$ be a collection of strings on an alphabet of σ letters. The sum of lengths of S_i is N.

- $GSA[i]$: The i-th smallest suffix of the strings in S. If $GSA[i] = (t, h)$, then it corresponds to the suffix starting at the position t of the string S_h.
- $EBWT[i]$: The symbol that (circularly) precedes the first symbol of the suffix of S_h.
- $LCP[i]$: The length of longest common prefix with preceding suffix in the sorted list of the suffixes of S.

Example

Multiset S

	0	1	2	3	4	5	6
S_1	G	C	C	A	A	C	$\$_1$
S_2	G	A	G	C	T	C	$\$_2$
S_3	T	C	G	C	T	T	$\$_3$

	GSA	LCP	$EBWT$	Sorted Suffixes of S
0	$(6,1)$	0	C	$\$_1$
1	$(6,2)$	0	C	$\$_2$
2	$(6,3)$	0	T	$\$_3$
3	$(3,1)$	0	C	$AAC\$_1$
4	$(4,1)$	1	A	$AC\$_1$
5	$(1,2)$	1	G	$AGCTC\$_2$
6	$(5,1)$	0	A	$C\$_1$
7	$(5,2)$	1	T	$C\$_2$
8	$(2,1)$	1	C	$CAAC\$_1$
9	$(1,1)$	1	G	$CCAAC\$_1$
10	$(1,3)$	1	T	$CGCTT\$_3$
11	$(3,2)$	1	G	$CTC\$_2$
12	$(3,3)$	2	G	$CTT\$_3$
13	$(0,2)$	0	$\$_2$	$GAGCTC\$_2$
14	$(0,1)$	1	$\$_1$	$GCCAAC\$_1$
15	$(2,2)$	2	A	$GCTC\$_2$
16	$(2,3)$	3	C	$GCTT\$_3$
17	$(5,3)$	0	T	$T\$_3$
18	$(4,2)$	1	C	$TC\$_2$
19	$(0,3)$	2	$\$_3$	$TCGCTT\$_3$
20	$(4,3)$	1	C	$TT\$_3$

Let S $= \{S_1, S_2, \ldots, S_m\}$ be a collection of strings on an alphabet of σ letters. The sum of lengths of S_i is N.

- $GSA[i]$: The i-th smallest suffix of the strings in S. If $GSA[i] = (t, h)$, then it corresponds to the suffix starting at the position t of the string S_h.
- $EBWT[i]$: The symbol that (circularly) precedes the first symbol of the suffix of S_h.
- $LCP[i]$: The length of longest common prefix with preceding suffix in the sorted list of the suffixes of S.

Example

Multiset S

	0	1	2	3	4	5	6
S_1	G	C	C	A	A	C	$\$_1$
S_2	G	A	G	C	T	C	$\$_2$
S_3	T	C	G	C	T	T	$\$_3$

	GSA	LCP	$EBWT$	Sorted Suffixes of S
0	$(6,1)$	0	C	$\$_1$
1	$(6,2)$	0	C	$\$_2$
2	$(6,3)$	0	T	$\$_3$
3	$(3,1)$	0	C	$AAC\$_1$
4	$(4,1)$	1	A	$AC\$_1$
5	$(1,2)$	1	G	$AGCTC\$_2$
6	$(5,1)$	0	A	$C\$_1$
7	$(5,2)$	1	T	$C\$_2$
8	$(2,1)$	1	C	$CAAC\$_1$
9	$(1,1)$	1	G	$CCAAC\$_1$
10	$(1,3)$	1	T	$CGCTT\$_3$
11	$(3,2)$	1	G	$CTC\$_2$
12	$(3,3)$	2	G	$CTT\$_3$
13	$(0,2)$	0	$\$_2$	$GAGCTC\$_2$
14	$(0,1)$	1	$\$_1$	$GCCAAC\$_1$
15	$(2,2)$	2	A	$GCTC\$_2$
16	$(2,3)$	3	C	$GCTT\$_3$
17	$(5,3)$	0	T	$T\$_3$
18	$(4,2)$	1	C	$TC\$_2$
19	$(0,3)$	2	$\$_3$	$TCGCTT\$_3$
20	$(4,3)$	1	C	$TT\$_3$

Experiments

instance	size in Gb	program	wall clock	efficiency	memory
0043M	4.00	BCR	0.99	0.84	0.57
	4.00	extLCP	3.29	0.98	1.00
0085M	8.00	BCR	1.01	0.83	1.10
	8.00	extLCP	3.81	0.87	2.00
0100M	9.31	BCR	1.05	0.81	1.35
	9.31	extLCP	4.03	0.83	2.30
0200M	18.62	BCR	1.63	0.58	4.00
	18.62	extLCP	4.28	0.79	4.70
0800M	74.51	BCR	3.23	0.43	10.40
	74.51	extLCP	6.68	0.67	18.00

- All reads are 100 bases long.
- wall clock time (the amount of time that elapsed from the start to the completion of the instance) is given as microseconds per input base.
- memory denotes the maximal amount of memory (in gigabytes) used during execution.
- The efficiency column states the CPU efficiency values, i.e. the proportion of time for which the CPU was occupied and not waiting for I/O operations to finish, as taken from the output of the `/usr/bin/time` command.

The `extLCP` algorithm:

- uses $O(mk^2 \log \sigma)$ disk I/O and $O((m+\sigma^2)\log(mk))$ bits of memory.
- takes $O(k(m+\mathrm{sort}(m)))$ CPU time, where $\mathrm{sort}(m)$ is the time taken to sort m integers.

BWT-based Compressors of a text

- BWT is a compression booster: BW-transformed text is compressed by chaining standard compression techniques.
- Once generated, the BWT is compressed by standard techniques: a typical scheme would follow an initial move-to-front encoding with run length encoding and then Huffman encoding.

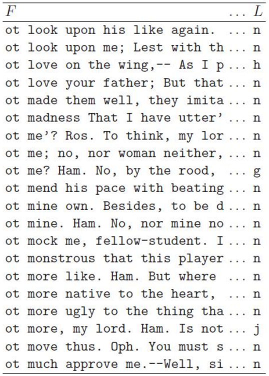
$$\begin{tikzcd} v \arrow[r] & \boxed{BWT} \arrow[r] & bwt(v) \arrow[r] & \boxed{\text{Compressor}} \arrow[r] & \text{Output} \end{tikzcd}$$

- For instance bzip2 (http://www.bzip.org, Julian Seward)
  - divides a text into blocks of (at most, and by default) 900 kB,
  - compresses each separately,
  - hence is only able to take advantage of local similarities in the data.

Why Useful?

INTUITION

Let us consider the effect of BWT on a segment of a BWT-sorted file for Shakespeare's Hamlet.



F	$\ldots L$
ot look upon his like again.	... n
ot look upon me; Lest with th	... n
ot love on the wing,-- As I p	... h
ot love your father; But that	... n
ot made them well, they imita	... n
ot madness That I have utter'	... n
ot me'? Ros. To think, my lor	... n
ot me; no, nor woman neither,	... n
ot me? Ham. No, by the rood,	... g
ot mend his pace with beating	... n
ot mine own. Besides, to be d	... n
ot mine. Ham. No, nor mine no	... n
ot mock me, fellow-student. I	... n
ot monstrous that this player	... n
ot more like. Ham. But where	... n
ot more native to the heart,	... n
ot more ugly to the thing tha	... n
ot more, my lord. Ham. Is not	... j
ot move thus. Oph. You must s	... n
ot much approve me.--Well, si	... n

The factor *ot* is normally preceded by *n*, but occasionally by *h*, *g* or *j*.

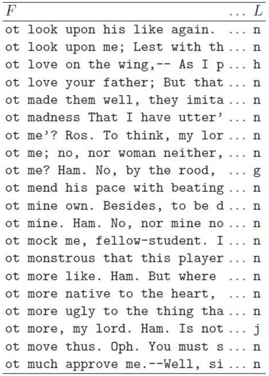
The characters preceding *ot* are grouped together.

The "clustering effect" is also kept when the Extended Burrows-Wheeler transform is used.

Why Useful?

INTUITION

Let us consider the effect of BWT on a segment of a BWT-sorted file for Shakespeare's Hamlet.



F	$\dots L$
ot look upon his like again.	... n
ot look upon me; Lest with th	... n
ot love on the wing,-- As I p	... h
ot love your father; But that	... n
ot made them well, they imita	... n
ot madness That I have utter'	... n
ot me'? Ros. To think, my lor	... n
ot me; no, nor woman neither,	... n
ot me? Ham. No, by the rood,	... g
ot mend his pace with beating	... n
ot mine own. Besides, to be d	... n
ot mine. Ham. No, nor mine no	... n
ot mock me, fellow-student. I	... n
ot monstrous that this player	... n
ot more like. Ham. But where	... n
ot more native to the heart,	... n
ot more ugly to the thing tha	... n
ot more, my lord. Ham. Is not	... j
ot move thus. Oph. You must s	... n
ot much approve me.--Well, si	... n

The factor *ot* is normally preceded by *n*, but occasionally by *h*, *g* or *j*.

The characters preceding *ot* are grouped together.

The "clustering effect" is also kept when the Extended Burrows-Wheeler transform is used.

BWT-based Compressors of a collection

Extended Goal

The EBWT-based Compressors of very large collections.

Let S $= \{S_1, S_2, \ldots, S_m\}$ be a collection of m strings.

- We use EBWT [Bauer, Cox and R., 2013] that works in external memory and compute the EBWT by sorting the suffixes of very large collections.
- Recall that EBWT requires ordered and distinct "end-marker" characters to be appended to the sequences.
- So, we assume that we use implicit distinct end markers, i.e. we suppose that

$$\$_1 = \$_2 = \ldots = \$_m = \$.$$

In the sense that we use the positions of the sequences in the multiset in order to establish the order relation between two identical suffixes.

BWT-based Compressors of a collection

Extended Goal

The EBWT-based Compressors of very large collections.

Let S $= \{S_1, S_2, \ldots, S_m\}$ be a collection of m strings.

- We use EBWT [Bauer, Cox and R., 2013] that works in external memory and compute the EBWT by sorting the suffixes of very large collections.
- Recall that EBWT requires ordered and distinct "end-marker" characters to be appended to the sequences.
- So, we assume that we use implicit distinct end markers, i.e. we suppose that

$$\$_1 = \$_2 = \ldots = \$_m = \$.$$

In the sense that we use the positions of the sequences in the multiset in order to establish the order relation between two identical suffixes.

BWT-based Compressors of a collection

Extended Goal

The EBWT-based Compressors of very large collections.

Let S $= \{S_1, S_2, \ldots, S_m\}$ be a collection of m strings.

- We use EBWT [Bauer, Cox and R., 2013] that works in external memory and compute the EBWT by sorting the suffixes of very large collections.
- Recall that EBWT requires ordered and distinct "end-marker" characters to be appended to the sequences.
- So, we assume that we use implicit distinct end markers, i.e. we suppose that

$$\$_1 = \$_2 = \ldots = \$_m = \$.$$

In the sense that we use the positions of the sequences in the multiset in order to establish the order relation between two identical suffixes.

BWT-based Compressors of a collection

Extended Goal

The EBWT-based Compressors of very large collections.

Let S $= \{S_1, S_2, \ldots, S_m\}$ be a collection of m strings.

- We use EBWT [Bauer, Cox and R., 2013] that works in external memory and compute the EBWT by sorting the suffixes of very large collections.
- Recall that EBWT requires ordered and distinct “end-marker” characters to be appended to the sequences.
- So, we assume that we use implicit distinct end markers, i.e. we suppose that

$$\$_1 = \$_2 = \ldots = \$_m = \$.$$

  In the sense that we use the positions of the sequences in the multiset in order to establish the order relation between two identical suffixes.

BWT-based Compressors of a collection

Extended Goal

The EBWT-based Compressors of very large collections.

Let S $= \{S_1, S_2, \ldots, S_m\}$ be a collection of m strings.

- We use EBWT [Bauer, Cox and R., 2013] that works in external memory and compute the EBWT by sorting the suffixes of very large collections.
- Recall that EBWT requires ordered and distinct "end-marker" characters to be appended to the sequences.
- So, we assume that we use implicit distinct end markers, i.e. we suppose that

  $$\$_1 = \$_2 = \ldots = \$_m = \$.$$

  In the sense that we use the positions of the sequences in the multiset in order to establish the order relation between two identical suffixes.

The use of end-markers

The use of ordered and (implicit or explicit) distinct "end-marker" symbols makes the multiset of sequences an ordered collection.

Problem

The use of the (implicit or explicit) distinct end-markers can affect the compression, since the same or similar sequences might be distant in the collection.

This can make the difference in the clustering effect!!!

Recall that the EBWT, defined in [Mantaci, Restivo, R. and Sciortino, 2005] does not require any end-marker.

A study of the combinatorial aspects that connect the $\preceq_\omega$-order among conjugates and the lexicographic order among suffixes of a multiset of words can be found in [Bonomo, Mantaci, Restivo, R. and Sciortino, 2013]. An important role is played by the notion of Lyndon word.

The use of end-markers

The use of ordered and (implicit or explicit) distinct "end-marker" symbols makes the multiset of sequences an ordered collection.

Problem

The use of the (implicit or explicit) distinct end-markers can affect the compression, since the same or similar sequences might be distant in the collection.

This can make the difference in the clustering effect!!!

Recall that the EBWT, defined in [Mantaci, Restivo, R. and Sciortino, 2005] does not require any end-marker.

A study of the combinatorial aspects that connect the $\preceq_\omega$-order among conjugates and the lexicographic order among suffixes of a multiset of words can be found in [Bonomo, Mantaci, Restivo, R. and Sciortino, 2013]. An important role is played by the notion of Lyndon word.

The use of end-markers

The use of ordered and (implicit or explicit) distinct “end-marker” symbols makes the multiset of sequences an ordered collection.

Problem

The use of the (implicit or explicit) distinct end-markers can affect the compression, since the same or similar sequences might be distant in the collection.

This can make the difference in the clustering effect!!!

Recall that the EBWT, defined in [Mantaci, Restivo, R. and Sciortino, 2005] does not require any end-marker.

A study of the combinatorial aspects that connect the $\preceq_\omega$-order among conjugates and the lexicographic order among suffixes of a multiset of words can be found in [Bonomo, Mantaci, Restivo, R. and Sciortino, 2013]. An important role is played by the notion of Lyndon word.

The use of end-markers

The use of ordered and (implicit or explicit) distinct “end-marker” symbols makes the multiset of sequences an ordered collection.

Problem

The use of the (implicit or explicit) distinct end-markers can affect the compression, since the same or similar sequences might be distant in the collection.

This can make the difference in the clustering effect!!!

Recall that the EBWT, defined in [Mantaci, Restivo, R. and Sciortino, 2005] does not require any end-marker.
A study of the combinatorial aspects that connect the $\preceq_{\omega}$-order among conjugates and the lexicographic order among suffixes of a multiset of words can be found in [Bonomo, Mantaci, Restivo, R. and Sciortino, 2013]. An important role is played by the notion of Lyndon word.

Example

Ordered collection: S $= \{TAGACCT, TACCACT, GAGACCT\}$

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

We use implicit distinct end markers, i.e. $\$_1 = \$_2 = \$_3 = \$$.
In particular, if the strings have the length k, we have $S_i[k] = S_j[k] = \$$, and we define $S_i[k] < S_j[k]$, if $i < j$.

Note that, we have a $1-1$ correspondence between symbols in EBWT and sorted list of all suffixes in the collection.

$ebwt(\mathsf{S}) = TTTTGGCTGCAAACACAA\$CCC\$\$$

Example

Ordered collection: S $= \{TAGACCT, TACCACT, GAGACCT\}$

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

We use implicit distinct end markers, i.e. $\$_1 = \$_2 = \$_3 = \$$.
In particular, if the strings have the length k, we have $S_i[k] = S_j[k] = \$$, and we define $S_i[k] < S_j[k]$, if $i < j$.

Note that, we have a $1-1$ correspondence between symbols in EBWT and sorted list of all suffixes in the collection.

$ebwt(\mathsf{S}) = TTTTGGCTGCAAACACAA\$CCC\$\$$

Example

Ordered collection: $\mathsf{S} = \{TAGACCT, TACCACT, GAGACCT\}$

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

We use implicit distinct end markers, i.e. $\$_1 = \$_2 = \$_3 = \$$.
In particular, if the strings have the length k, we have $S_i[k] = S_j[k] = \$$, and we define $S_i[k] < S_j[k]$, if $i < j$.

Note that, we have a $1-1$ correspondence between symbols in EBWT and sorted list of all suffixes in the collection.

$ebwt(\mathsf{S}) = TTTTGGCTGCAAACACAA\$CCC\$\$$

Example

Ordered collection: $S = \{TAGACCT, TACCACT, GAGACCT\}$

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

We use implicit distinct end markers, i.e. $\$_1 = \$_2 = \$_3 = \$$. In particular, if the strings have the length k, we have $S_i[k] = S_j[k] = \$$, and we define $S_i[k] < S_j[k]$, if $i < j$.

Note that, we have a $1-1$ correspondence between symbols in EBWT and sorted list of all suffixes in the collection.

$$ebwt(S) = TTTTGGCTGCAAACACAA\$CCC\$\$$$

Example: swapping sequences

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	$T\$$
$\$$	$TACCACT\$$
$\$$	$TAGACCT\$$

Example: swapping sequences

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	$T\$$
$\$$	$TACCACT\$$
$\$$	$TAGACCT\$$

Example: swapping sequences

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

$EBWT$	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	$T\$$
$\$$	$TACCACT\$$
$\$$	$TAGACCT\$$

Example: swapping sequences

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	$T\$$
$\$$	$TACCACT\$$
$\$$	$TAGACCT\$$

Reordering of the sequences [Cox, Bauer, Jakobi and R., 2012]

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

$EBWT$	$Suffixes$
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	$T\$$
$\$$	$TACCACT\$$
$\$$	$TAGACCT\$$

Key insight

In these regions, when the non-$ suffixes are the same, the ordering is determined by the ordering of the reads in the collection.

Idea

Change the ordering of the reads to get a better compression in these regions.

- If we swap $TAGACCT$ and $TACCACT$ in the ordered multiset,
- we should swap the symbols C and A in the EBWT,
- then we could obtain a better compression.

Reordering of the sequences [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	$T\$$
$\$$	$TACCACT\$$
$\$$	$TAGACCT\$$

Key insight

In these regions, when the non-$\$$ suffixes are the same, the ordering is determined by the ordering of the reads in the collection.

Idea

Change the ordering of the reads to get a better compression in these regions.

- If we swap $TAGACCT$ and $TACCACT$ in the ordered multiset,
- we should swap the symbols C and A in the EBWT,
- then we could obtain a better compression.

Reordering of the sequences [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	$T\$$
$\$$	$TACCACT\$$
$\$$	$TAGACCT\$$

Key insight

In these regions, when the non-$\$$ suffixes are the same, the ordering is determined by the ordering of the reads in the collection.

Idea

Change the ordering of the reads to get a better compression in these regions.

- If we swap $TAGACCT$ and $TACCACT$ in the ordered multiset,
- we should swap the symbols C and A in the EBWT,
- then we could obtain a better compression.

Reordering of the sequences [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	$T\$$
$\$$	$TACCACT\$$
$\$$	$TAGACCT\$$

Key insight

In these regions, when the non-\$ suffixes are the same, the ordering is determined by the ordering of the reads in the collection.

Idea

Change the ordering of the reads to get a better compression in these regions.

- If we swap $TAGACCT$ and $TACCACT$ in the ordered multiset,
- we should swap the symbols C and A in the EBWT,
- then we could obtain a better compression.

Reordering of the sequences [Cox, Bauer, Jakobi and R., 2012]

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

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

Key insight

In these regions, when the non-$ suffixes are the same, the ordering is determined by the ordering of the reads in the collection.

Idea

Change the ordering of the reads to get a better compression in these regions.

- If we swap $TAGACCT$ and $TACCACT$ in the ordered multiset,
- we should swap the symbols C and A in the EBWT,
- then we could obtain a better compression.

Reordering of the sequences [Cox, Bauer, Jakobi and R., 2012]

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

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

Key insight

In these regions, when the non-\$ suffixes are the same, the ordering is determined by the ordering of the reads in the collection.

Idea

Change the ordering of the reads to get a better compression in these regions.

- If we swap $TAGACCT$ and $TACCACT$ in the ordered multiset,
- we should swap the symbols C and A in the EBWT,
- then we could obtain a better compression.

Reordering of the sequences [Cox, Bauer, Jakobi and R., 2012].

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

- So, by swapping $TAGA\underline{C}CT$ with $TACC\underline{A}CT$, the initial ordered collection:

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

becomes:

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

- Now the Cs associated with the suffixes $CT\$$ are adjacent.
- Rest of EBWT is unaffected by this change in ordering.

Reordering of the sequences [Cox, Bauer, Jakobi and R., 2012].

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

- So, by swapping $TAGA\underline{C}CT$ with $TACC\underline{A}CT$, the initial ordered collection:

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

  becomes:

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

- Now the Cs associated with the suffixes $CT\$$ are adjacent.
- Rest of EBWT is unaffected by this change in ordering.

Reordering of the sequences [Cox, Bauer, Jakobi and R., 2012].

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

- So, by swapping $TAGA\underline{C}CT$ with $TACC\underline{A}CT$, the initial ordered collection:

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

becomes:

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

- Now the Cs associated with the suffixes $CT\$$ are adjacent.
- Rest of EBWT is unaffected by this change in ordering.

How to do this reordering? [Cox, Bauer, Jakobi and R., 2012].

The initial ordered collection: S $= \{TAGA\underline{C}CT, TACC\underline{A}CT, GAGACCT\}$
becomes: S $= \{TACC\underline{A}CT, TAGA\underline{C}CT, GAGACCT\}$

$EBWT$	$Suffixes$
$T$	\$
$T$	\$
$T$	\$
$T$	$ACCACT\$$
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	$T\$$
\$	$TACCACT\$$
\$	$TAGACCT\$$

Strategy RLO pre-sort reads into reverse lexicographic order. This ensures EBWT symbols associated with such suffixes are grouped together.

Strategy SAP modify EBWT construction algorithm to add extra bit that tracks whether each suffix is "Same As Previous". Minimal additional overhead. Then make a single pass through the EBWT to do the grouping.

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

How to do this reordering? [Cox, Bauer, Jakobi and R., 2012].

The initial ordered collection: S $= \{TAGA\underline{C}CT, TACC\underline{A}CT, GAGACCT\}$
becomes: S $= \{TACC\underline{A}CT, TAGA\underline{C}CT, GAGACCT\}$

$EBWT$	$Suffixes$
$T$	$\$$
$T$	$\$$
$T$	$\$$
$T$	$ACCACT\$$
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	$T\$$
$\$$	$TACCACT\$$
$\$$	$TAGACCT\$$

Strategy RLO pre-sort reads into reverse lexicographic order. This ensures EBWT symbols associated with such suffixes are grouped together.

Strategy SAP modify EBWT construction algorithm to add extra bit that tracks whether each suffix is "Same As Previous". Minimal additional overhead. Then make a single pass through the EBWT to do the grouping.

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

How to do this reordering? [Cox, Bauer, Jakobi and R., 2012].

The initial ordered collection: S $= \{TAGA\underline{C}CT, TACC\underline{A}CT, GAGACCT\}$
becomes: S $= \{TACC\underline{A}CT, TAGA\underline{C}CT, GAGACCT\}$

$EBWT$	$Suffixes$
$T$	\$
$T$	\$
$T$	\$
$T$	$ACCACT\$$
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	$T\$$
\$	$TACCACT\$$
\$	$TAGACCT\$$

Strategy RLO pre-sort reads into reverse lexicographic order. This ensures EBWT symbols associated with such suffixes are grouped together.

Strategy SAP modify EBWT construction algorithm to add extra bit that tracks whether each suffix is "Same As Previous". Minimal additional overhead. Then make a single pass through the EBWT to do the grouping.
Outcome is EBWT of a permuted read collection. Can verify by inverting the EBWT.

Experiments

Method		Time		Compression
Stage 1	Stage 2	Stage 1	Stage 2	bits per base
Reads	Bzip2	-	905	2.25
	PPMd (default)		324	2.04
	PPMd (large)		5155	2.00
	-mx9		17974	1.98
EBWT	Bzip2	3520	818	2.09
	PPMd (default)		353	1.93
	PPMd (large)		4953	2.05
	-mx9		16709	2.09
EBWT-SAP	Bzip2	3520	601	1.40
	PPMd (default)		347	1.21
	PPMd (large)		3116	1.28
	-mx9		11204	1.34

Different combinations of first-stage (EBWT, SAP-permuted EBWT) and second-stage (bzip2 with default parameters, PPMd mode of `7-Zip` with default parameters, PPMd mode of `7-Zip` with `-mo=16 -mmem=2048m`, deflate mode of `7-Zip` with `-mx9`) compression compared on 192 million human reads previous analyzed by[Yanovsky, 2011]. Time is in CPU seconds, as measured on a single core of an Intel Xeon X5450 (Quad-core) 3GHz processor.

Compression of quality scores

Goal

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

Recall that

- Quality scores are assigned to each nucleotide base call in sequencer.
- Typically quality score is an integer that expresses error probability on the Phred scale

$$Q_{phred} = -10\log_{10}p$$

where p is the error probability.
- Phred quality scores have become widely accepted to characterize the quality of DNA sequences, and can be used to compare the efficacy of different sequencing methods.
- the quality scores could require more space than the sequences themselves.

Compression of quality scores

Goal

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

Recall that

- Quality scores are assigned to each nucleotide base call in sequencer.
- Typically quality score is an integer that expresses error probability on the Phred scale

$$Q_{phred} = -10\log_{10}p$$

where p is the error probability.

- Phred quality scores have become widely accepted to characterize the quality of DNA sequences, and can be used to compare the efficacy of different sequencing methods.
- the quality scores could require more space than the sequences themselves.

Compression of quality scores

Goal

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

Recall that

- Quality scores are assigned to each nucleotide base call in sequencer.
- Typically quality score is an integer that expresses error probability on the Phred scale
  $$Q_{phred} = -10\log_{10}p$$
  where p is the error probability.
- Phred quality scores have become widely accepted to characterize the quality of DNA sequences, and can be used to compare the efficacy of different sequencing methods.
- the quality scores could require more space than the sequences themselves.

Compression of quality scores

Goal

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

Recall that

- Quality scores are assigned to each nucleotide base call in sequencer.
- Typically quality score is an integer that expresses error probability on the Phred scale

$$Q_{phred} = -10\log_{10}p$$

where p is the error probability.

- Phred quality scores have become widely accepted to characterize the quality of DNA sequences, and can be used to compare the efficacy of different sequencing methods.
- the quality scores could require more space than the sequences themselves.

Compression of quality scores

Goal

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

Recall that

- Quality scores are assigned to each nucleotide base call in sequencer.
- Typically quality score is an integer that expresses error probability on the Phred scale

$$Q_{phred} = -10\log_{10}p$$

  where p is the error probability.
- Phred quality scores have become widely accepted to characterize the quality of DNA sequences, and can be used to compare the efficacy of different sequencing methods.
- the quality scores could require more space than the sequences themselves.

Compression of quality scores

Goal

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

Recall that

- Quality scores are assigned to each nucleotide base call in sequencer.
- Typically quality score is an integer that expresses error probability on the Phred scale

$$Q_{phred} = -10\log_{10}p$$

where p is the error probability.
- Phred quality scores have become widely accepted to characterize the quality of DNA sequences, and can be used to compare the efficacy of different sequencing methods.
- the quality scores could require more space than the sequences themselves.

Adaptive compression of quality scores

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 compression of quality scores

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 compression of quality scores

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 compression of quality scores

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, 2013]

$Genoma$

$PEACHxBANANAxAPPLExPEARxTANGERINExORANGExPEACHxBANANAxPEAR$

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 $BANANA$.
- Letters 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 $PEAR$.
- Letters 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, 2013]

$$Genoma$$
$$PEACHxBANANAxAPPLExPEARxTANGERINExORANGExPEACHxBANANAxPEAR$$

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 $BANANA$.
- Letters 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 $PEAR$.
- Letters 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, 2013]

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*.
- Letters 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*.
- Letters 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, 2013]

$$\begin{array}{c} Genoma \\ PEACHxBANANAxAPPLExPEARxTANGERINExORANGExPEACHxBANANAxPEAR \end{array}$$

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 $BANANA$.
- Letters 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 $PEAR$.
- Letters 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, 2013]

$Genoma$

$PEACHxBANANAxAPPLExPEARxTANGERINExORANGExPEACHxBANANAxPEAR$

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 $BANANA$.
- Letters 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 $PEAR$.
- Letters 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, 2013]

$Genoma$

$PEACHxBANANAxAPPLExPEARxTANGERINExORANGExPEACHxBANANAxPEAR$

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 $BANANA$.
- Letters 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 $PEAR$.
- Letters 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, 2013]

$$Genoma$$

$$PEACHxBANANAxAPPLExPEARxTANGERINExORANGExPEACHxBANANAxPEAR$$

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 $BANANA$.
- Letters 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 $PEAR$.
- Letters 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, 2013]

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 $BANANA$.
- Letters 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 $PEAR$.
- Letters 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, 2013]

$Genoma$

$PEACHxBANANAxAPPLExPEARxTANGERINExORANGExPEACHxBANANAxPEAR$

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 $BANANA$.
- Letters 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 $PEAR$.
- Letters 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.

Smoothing quality scores

We use The EBWT and the LCP ("longest-common-prefix") array of the reads [Bauer, Cox, R. and Sciortino, 2012];
And we use LCP-array to define "LCP-intervals" (see [Abouelhoda Kurtz and Ohlebusch, 2004]).

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

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.

Smoothing quality scores

We use The EBWT and the LCP ("longest-common-prefix") array of the reads [Bauer, Cox, R. and Sciortino, 2012];
And we use LCP-array to define "LCP-intervals" (see [Abouelhoda Kurtz and Ohlebusch, 2004]).

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

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?

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

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).

Experiments

- Data: 33-fold coverage of C.elegans, 100-mer single reads: 33.808.546 reads of length 100.
- Set $c = 5$, $s = 10$: 76.8% of scores are smoothed
- Scores compressed using PPMd mode of 7-zip

  Original scores: 2.51 bits/score in EBWT space (FASTQ)
  Smoothed scores: 1.28 bits/score in EBWT space (FASTQ)

How to smooth?

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

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).

Experiments

- Data: 33-fold coverage of C.elegans, 100-mer single reads: $33.808.546$ reads of length 100.
- Set $c = 5$, $s = 10$: 76.8% of scores are smoothed
- Scores compressed using PPMd mode of 7-zip
  - Original scores: **2.51** bits/score in EBWT space FASTQ);
  - Smoothed scores: **1.28** bits/score in EBWT space FASTQ).

How to smooth?

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

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).

Experiments

- Data: 33-fold coverage of C.elegans, 100-mer single reads: $33.808.546$ reads of length 100.
- Set $c = 5$, $s = 10$: 76.8% of scores are smoothed
- Scores compressed using PPMd mode of 7-zip
  - Original scores: **2.51** bits/score in EBWT space FASTQ);
  - Smoothed scores: **1.28** bits/score in EBWT space FASTQ).

How to smooth?

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

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).

Experiments

- Data: 33-fold coverage of C.elegans, 100-mer single reads: $33.808.546$ reads of length 100.
- Set $c = 5$, $s = 10$: 76.8% of scores are smoothed
- Scores compressed using PPMd mode of 7-zip
  - Original scores: **2.51** bits/score in EBWT space FASTQ);
  - Smoothed scores: **1.28** bits/score in EBWT space FASTQ).

How to smooth?

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

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).

Experiments

- Data: 33-fold coverage of C.elegans, 100-mer single reads: $33.808.546$ reads of length 100.
- Set $c = 5$, $s = 10$: 76.8% of scores are smoothed
- Scores compressed using PPMd mode of 7-zip
  - Original scores: **2.51** bits/score in EBWT space FASTQ);
  - Smoothed scores: **1.28** bits/score in EBWT space FASTQ).

Adaptive reference-free compression

- Have given a reference-free and “intelligently lossy” approach to quality score smoothing.
- Only keep scores for bases that are likely to be important downstream.
- Our smoothing strategy is simplest possible (symbols preceding a context must agree unanimously)
- but this work provides framework for analysing more sophisticated approaches.

Comparing DNA Sequence Collections [Cox, Jakobi and R., 2012]

Task

Given EBWTs of two sets of reads R and G, find all k-mers that are

- Present in R only;
- Present in G only;
- Present in both R and G.

- We do this by making k sequential passes through EBWT of G and EBWT of R.
- We can do this by using sequential access (can read files from disk, no RAM needed).

Key idea

All-against-all backward search in external memory

Applications: Finding splice junctions without a reference.

Comparing DNA Sequence Collections [Cox, Jakobi and R., 2012]

Task

Given EBWTs of two sets of reads R and G, find all k-mers that are

- Present in R only;
- Present in G only;
- Present in both R and G.

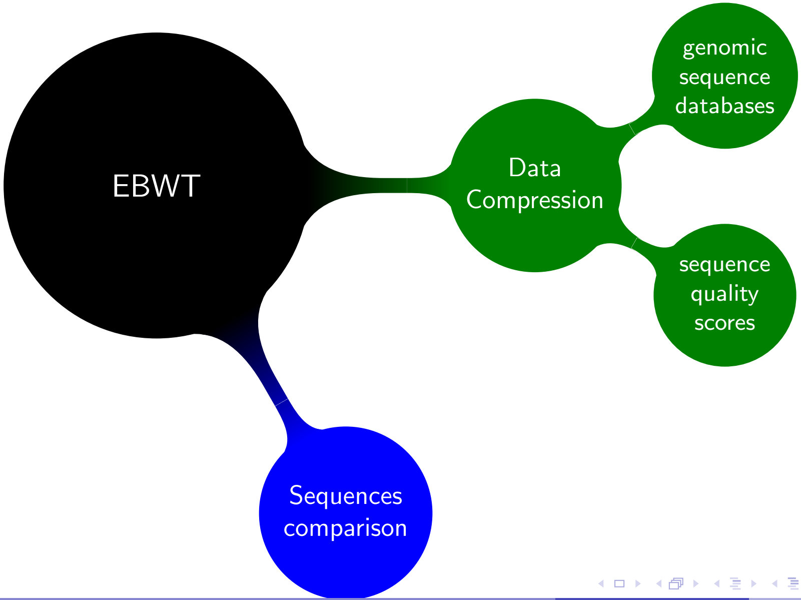
- We do this by making k sequential passes through EBWT of G and EBWT of R.
- We can do this by using sequential access (can read files from disk, no RAM needed).

Key idea

All-against-all backward search in external memory

Applications: Finding splice junctions without a reference.

Conclusions: EBWT as tool



References I

Abouelhoda, M. I., Kurtz, S., and Ohlebusch, E. (2004).
Replacing suffix trees with enhanced suffix arrays.
Journal of Discrete Algorithms, 2(1):53 – 86.

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

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

References II

Bauer, M. J., Cox, A. J., Rosone, G., and Sciortino, M. (2012).
Lightweight LCP construction for next-generation sequencing datasets.
In *WABI*, volume 7534 LNBI of *LNCS*, pages 326–337. Springer.

Bonomo, S., Mantaci, S., Restivo, A., Rosone, G., and Sciortino, M. (2013).
Suffixes, Conjugates and Lyndon words.
In *DLT*, volume 7907 of *LNCS*, pages 131–142. Springer.

Cox, A. J., Bauer, M. J., Jakobi, T., and Rosone, G. (2012a).
Large-scale compression of genomic sequence databases with the Burrows-Wheeler transform.
Bioinformatics, 28(11):1415–1419.

References III

Cox, A. J., Jakobi, T., Rosone, G., and Schulz-Trieglaff, O. B. (2012b).
Comparing DNA sequence collections by direct comparison of compressed text indexes.
In *WABI*, volume 7534 LNBI of *LNCS*, pages 214–224. Springer.

Ferragina, P., Gagie, T., and Manzini, G. (2012).
Lightweight Data Indexing and Compression in External Memory.
Algorithmica, 63(3):707–730.

Hon, W.-K., Ku, T.-H., Lu, C.-H., Shah, R., and Thankachan, S. V. (2012).
Efficient Algorithm for Circular Burrows-Wheeler Transform.
In *CPM*, volume 7354 of *LNCS*, pages 257–268. Springer.

References IV

Janin, L., Rosone, G., and Cox, A. J. (First published online May 9, 2013).
Adaptive reference-free compression of sequence quality scores.
Bioinformatics.

Mantaci, S., Restivo, A., Rosone, G., and Sciortino, M. (2005).
An Extension of the Burrows Wheeler Transform and Applications to Sequence Comparison and Data Compression.
In *CPM*, volume 3537 of *LNCS*, pages 178–189. Springer.

Mantaci, S., Restivo, A., Rosone, G., and Sciortino, M. (2007).
An extension of the Burrows-Wheeler Transform.
Theoret. Comput. Sci., 387(3):298–312.

References V

Sirén, J. (2009).
Compressed suffix arrays for massive data.
In *SPIRE*, volume 5721 of *LNCS*, pages 63–74. Springer.

Yanovsky, V. (2011).
ReCoil - an algorithm for compression of extremely large datasets of DNA data.
Algorithms for Molecular Biology, 6(1):23.

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

github.com:BEETL/BEETL.git

Thank you for your attention!