

# Lightweight BWT Construction for Very Large String Collections

Markus J. Bauer, Anthony J. Cox and Giovanna Rosone

Computational Biology Group, Illumina Cambridge Ltd., United Kingdom  
Dipartimento di Matematica e Informatica, University of Palermo, Palermo, ITALY

Workshop PRIN, 5-7 September 2011

# 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)
- Whole human genome sequencing: about 3Gbase genome typically sampled to 20 to 30-fold redundancy to ensure adequate coverage of both copies (i.e. each position in the genome sampled 30 times, on average)
- Datasets of 100 Gbases or more are common
- Applications: Comparing of genomes, assembl, alignment, ...

# The BWT

- The BWT is a reversible transformation that produces a permutation  $bwt(v)$  of an input sequence  $v$ , defined over an ordered alphabet  $\Sigma$ , so that occurrences of a given symbol tend to occur in clusters in the output sequence.
- Traditionally the major application of the Burrows-Wheeler Transform has been for Data Compression. The BWT represents for instance the heart of the BZIP2 algorithm.
- Today, there are reports of the application of the BWT in bio-informatics, full-text compressed indexes, prediction and entropy estimation, and shape analysis in computer vision, etc.

# The BWT

- The BWT is a reversible transformation that produces a permutation  $bwt(v)$  of an input sequence  $v$ , defined over an ordered alphabet  $\Sigma$ , so that occurrences of a given symbol tend to occur in clusters in the output sequence.
- Traditionally the major application of the Burrows-Wheeler Transform has been for Data Compression. The BWT represents for instance the heart of the BZIP2 algorithm.
- Today, there are reports of the application of the BWT in bio-informatics, full-text compressed indexes, prediction and entropy estimation, and shape analysis in computer vision, etc.

# The BWT

- The BWT is a reversible transformation that produces a permutation  $bwt(v)$  of an input sequence  $v$ , defined over an ordered alphabet  $\Sigma$ , so that occurrences of a given symbol tend to occur in clusters in the output sequence.
- Traditionally the major application of the Burrows-Wheeler Transform has been for Data Compression. The BWT represents for instance the heart of the BZIP2 algorithm.
- Today, there are reports of the application of the BWT in bio-informatics, full-text compressed indexes, prediction and entropy estimation, and shape analysis in computer vision, etc.

# How does BWT work?

- BWT takes as input a text  $v$ , append  $\$$  to the end of  $v$  ( $\$$  is unique and smaller than any other character) 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:  $v = abraca$

- Each row of  $M$  is a conjugate of  $v\$$  in lexicographic order.
- $bwt(v)$  coincides with the last column  $L$  of the BW-matrix  $M$ .
- The index  $I$  is the row of  $M$  containing the original sequence followed by  $\$$ .

		$M$						
	$F$						$L$	
	$\downarrow$						$\downarrow$	
	0	\$	a	b	r	a	c	a
	1	a	\$	a	b	r	a	c
$I \rightarrow$	2	a	b	r	a	c	a	\$
	3	a	c	a	\$	a	b	r
	4	b	r	a	c	a	\$	a
	5	c	a	\$	a	b	r	a
	6	r	a	c	a	\$	a	b

# How does BWT work?

- BWT takes as input a text  $v$ , append  $\$$  to the end of  $v$  ( $\$$  is unique and smaller than any other character) 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:  $v = abraca$

- Each row of  $M$  is a conjugate of  $v\$$  in lexicographic order.
- $bwt(v)$  coincides with the last column  $L$  of the BW-matrix  $M$ .
- The index  $I$  is the row of  $M$  containing the original sequence followed by  $\$$ .

		$M$						
	$F$						$L$	
	$\downarrow$						$\downarrow$	
	0	\$	a	b	r	a	c	a
	1	a	\$	a	b	r	a	c
$I \rightarrow$	2	a	b	r	a	c	a	\$
	3	a	c	a	\$	a	b	r
	4	b	r	a	c	a	\$	a
	5	c	a	\$	a	b	r	a
	6	r	a	c	a	\$	a	b

# How does BWT work?

- BWT takes as input a text  $v$ , append  $\$$  to the end of  $v$  ( $\$$  is unique and smaller than any other character) 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:  $v = abra\color{green}{ca}$

- Each row of  $M$  is a conjugate of  $v\$$  in lexicographic order.
- $bwt(v)$  coincides with the last column  $L$  of the BW-matrix  $M$ .
- The index  $I$  is the row of  $M$  containing the original sequence followed by  $\$$ .

		$M$						
	$F$						$L$	
	$\downarrow$						$\downarrow$	
	0	\$	a	b	r	a	c	a
	1	a	\$	a	b	r	a	c
$I \rightarrow$	2	a	b	r	a	c	a	\$
	3	a	c	a	\$	a	b	r
	4	b	r	a	c	a	\$	a
	5	c	a	\$	a	b	r	a
	6	r	a	c	a	\$	a	b



# Properties

The following properties hold:

- 1 For all  $i = 0, \dots, |v|$ ,  $i \neq I$ , the character  $F[i]$  follows  $L[i]$  in the original string;
- 2 for each character  $c$ , the  $r$ -th occurrence of  $c$  in  $F$  corresponds to the  $r$ -th occurrence of  $c$  in  $L$ .

Ferragina and Manzini (2000) noticed the following connection:

$$LF[i] = C[L[i]] + \text{rank}(L[i], i - 1) \quad M$$

For instance:

if  $i = 5$  then  $L[i] = a$  and

$$LF[5] = C[a] + \text{rank}(a, 4) = 1 + 2 = 3$$

	$F$						$L$
	$\downarrow$						$\downarrow$
0	\$	a	b	r	a	c	a
1	a	\$	a	b	r	a	c
2	a	b	r	a	c	a	\$
3	a	c	a	\$	a	b	r
4	b	r	a	c	a	\$	a
5	c	a	\$	a	b	r	a
6	r	a	c	a	\$	a	b

$I \rightarrow$

# Properties

The following properties hold:

- 1 For all  $i = 0, \dots, |v|$ ,  $i \neq I$ , the character  $F[i]$  follows  $L[i]$  in the original string;
- 2 for each character  $c$ , the  $r$ -th occurrence of  $c$  in  $F$  corresponds to the  $r$ -th occurrence of  $c$  in  $L$ .

Ferragina and Manzini (2000) noticed the following connection:

$$LF[i] = C[L[i]] + \text{rank}(L[i], i - 1) \quad M$$

For instance:

if  $i = 5$  then  $L[i] = a$  and

$$LF[5] = C[a] + \text{rank}(a, 4) = 1 + 2 = 3$$

	$F$						$L$
	↓						↓
0	\$	a	b	r	a	c	a
1	a	\$	a	b	r	a	c
$I \rightarrow$ 2	a	b	r	a	c	a	\$
3	a	c	a	\$	a	b	r
4	b	r	a	c	a	\$	a
5	c	a	\$	a	b	r	a
6	r	a	c	a	\$	a	b

# Properties

The following properties hold:

- 1 For all  $i = 0, \dots, |v|$ ,  $i \neq I$ , the character  $F[i]$  follows  $L[i]$  in the original string;
- 2 for each character  $c$ , the  $r$ -th occurrence of  $c$  in  $F$  corresponds to the  $r$ -th occurrence of  $c$  in  $L$ .

Ferragina and Manzini (2000) noticed the following connection:

$$LF[i] = C[L[i]] + \text{rank}(L[i], i - 1) \quad M$$

For instance:

if  $i = 5$  then  $L[i] = a$  and

$$LF[5] = C[a] + \text{rank}(a, 4) = 1 + 2 = 3$$

	$F$						$L$
	↓						↓
0	\$	a	b	r	a	c	a
1	a	\$	a	b	r	a	c
$I \rightarrow$ 2	a	b	r	a	c	a	\$
3	a	c	a	\$	a	b	r
4	b	r	a	c	a	\$	a
5	c	a	\$	a	b	r	a
6	r	a	c	a	\$	a	b

# Properties

The following properties hold:

- 1 For all  $i = 0, \dots, |v|$ ,  $i \neq I$ , the character  $F[i]$  follows  $L[i]$  in the original string;
- 2 for each character  $c$ , the  $r$ -th occurrence of  $c$  in  $F$  corresponds to the  $r$ -th occurrence of  $c$  in  $L$ .

Ferragina and Manzini (2000) noticed the following connection:

$$LF[i] = C[L[i]] + \text{rank}(L[i], i - 1) \quad M$$

For instance:

if  $i = 5$  then  $L[i] = a$  and

$$LF[5] = C[a] + \text{rank}(a, 4) = 1 + 2 = 3$$

	$F$						$L$
	$\downarrow$						$\downarrow$
0	\$	a	b	r	a	c	a
1	a	\$	a	b	r	a	c
$I \rightarrow 2$	a	b	r	a	c	a	\$
3	a	c	a	\$	a	b	r
4	b	r	a	c	a	\$	a
5	c	a	\$	a	b	r	a
6	r	a	c	a	\$	a	b

# The BWT in bioinformatics

- 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)
- Create index from reference genome (e.g. human)  
create once, use many times
- Simpson and Durbin, Bioinformatics 2010: FM-index of a set of DNA sequences for overlap detection stage of de novo assembly  
See also Valimaki et al., CPM 2010

# The BWT in bioinformatics

- 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)
- Create index from reference genome (e.g. human)  
create once, use many times
- Simpson and Durbin, Bioinformatics 2010: FM-index of a set of DNA sequences for overlap detection stage of de novo assembly  
See also Valimaki et al., CPM 2010

# The BWT in bioinformatics

- 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)
- Create index from reference genome (e.g. human)  
create once, use many times
- Simpson and Durbin, Bioinformatics 2010: FM-index of a set of DNA sequences for overlap detection stage of de novo assembly  
See also Valimaki et al., CPM 2010

# BWT of a collection of strings

- BWT extended to set of strings by S. Mantaci et al. (CPM 2005, TCS 2007) by using a different ordering of the conjugates of the strings.
- original BWT of concatenated strings

- Straightforward to compute BWT from suffix array.
- Lots of work on efficient linear time SA generation methods.

- But, need to hold SA in RAM (complex or AL estimate)  $\Rightarrow$  memory
- They called

“[Space-Saving BWT](#)” (also called [Space-Saving BWT](#))

“[Space-Saving BWT](#)” (also called [Space-Saving BWT](#))

“[Space-Saving BWT](#)”



# BWT of a collection of strings

- BWT extended to set of strings by S. Mantaci et al. (CPM 2005, TCS 2007) by using a different ordering of the conjugates of the strings.
- original BWT of concatenated strings
  - Straightforward to compute BWT from suffix array.
  - Lots of work on efficient linear time SA generation methods.
  - But: need to hold SA in RAM (Simpson et al. estimate 700Gbytes RAM for SA of 60 Gbases of data)
  - Other options:
    - Siren, SPIRE 2009: divide collection into batches, compute BWT of each then merge
    - Ferragina et al., Latin 2010: partition string  $T$  into blocks  $T_1, \dots, T_k$ , create SA of each in turn

# BWT of a collection of strings

- BWT extended to set of strings by S. Mantaci et al. (CPM 2005, TCS 2007) by using a different ordering of the conjugates of the strings.
- original BWT of concatenated strings
  - Straightforward to compute BWT from suffix array.
  - Lots of work on efficient linear time SA generation methods.
  - **But:** need to hold SA in RAM (Simpson et al. estimate 700Gbytes RAM for SA of 60 Gbases of data)
  - Other options:
    - Siren, SPIRE 2009: divide collection into batches, compute BWT of each then merge
    - Ferragina et al., Latin 2010: partition string  $T$  into blocks  $T_r \cdots T_1$ , create SA of each in turn

# BWT of a collection of strings

- BWT extended to set of strings by S. Mantaci et al. (CPM 2005, TCS 2007) by using a different ordering of the conjugates of the strings.
- original BWT of concatenated strings
  - Straightforward to compute BWT from suffix array.
  - Lots of work on efficient linear time SA generation methods.
  - **But:** need to hold SA in RAM (Simpson et al. estimate 700Gbytes RAM for SA of 60 Gbases of data)
  - Other options:
    - Siren, SPIRE 2009: divide collection into batches, compute BWT of each then merge
    - Ferragina et al., Latin 2010: partition string  $T$  into blocks  $T_r \cdots T_1$ , create SA of each in turn

# BWT of a collection of strings

- BWT extended to set of strings by S. Mantaci et al. (CPM 2005, TCS 2007) by using a different ordering of the conjugates of the strings.
- original BWT of concatenated strings
  - Straightforward to compute BWT from suffix array.
  - Lots of work on efficient linear time SA generation methods.
  - **But**: need to hold SA in RAM (Simpson et al. estimate 700Gbytes RAM for SA of 60 Gbases of data)
  - Other options:
    - Siren, SPIRE 2009: divide collection into batches, compute BWT of each then merge
    - Ferragina et al., Latin 2010: partition string  $T$  into blocks  $T_r \cdots T_1$ , create SA of each in turn

# Observations

Let  $S$  be a collection of  $m$  strings of length  $k$  on an alphabet of  $\sigma$  letters. Our algorithm computes the BWT of  $S$

- without concatenating the strings belonging to  $S$  and without needing to compute their suffix array.
- incrementally via  $k$  iterations. At each of the iterations  $j = 1, 2, \dots, k$ , the algorithm computes a partial BWT string  $\text{bwt}_j(S)$  by inserting the symbols preceding the  $j$ -suffixes of  $S$  at their correct positions into  $\text{bwt}_{j-1}(S)$ . Each iteration  $j$  simulates the insertion of the  $j$ -suffixes in the suffix array.
- The string  $\text{bwt}_j(S)$  is a 'partial BWT' in the sense that the addition of  $m$  end markers in their correct positions would make it the BWT of the collection  $\{S_1[k-j-1, k], S_2[k-j-1, k], \dots, S_m[k-j-1, k]\}$ .
- *This insertion does not affect the relative ordering of symbols inserted during previous iterations.*

# Observations

Let  $S$  be a collection of  $m$  strings of length  $k$  on an alphabet of  $\sigma$  letters. Our algorithm computes the BWT of  $S$

- without concatenating the strings belonging to  $S$  and without needing to compute their suffix array.
- incrementally via  $k$  iterations. At each of the iterations  $j = 1, 2, \dots, k$ , the algorithm computes a partial BWT string  $\text{bwt}_j(S)$  by inserting the symbols preceding the  $j$ -suffixes of  $S$  at their correct positions into  $\text{bwt}_{j-1}(S)$ . Each iteration  $j$  simulates the insertion of the  $j$ -suffixes in the suffix array.
- The string  $\text{bwt}_j(S)$  is a 'partial BWT' in the sense that the addition of  $m$  end markers in their correct positions would make it the BWT of the collection  $\{S_1[k-j-1, k], S_2[k-j-1, k], \dots, S_m[k-j-1, k]\}$ .
- *This insertion does not affect the relative ordering of symbols inserted during previous iterations.*

# Observations

Let  $S$  be a collection of  $m$  strings of length  $k$  on an alphabet of  $\sigma$  letters. Our algorithm computes the BWT of  $S$

- without concatenating the strings belonging to  $S$  and without needing to compute their suffix array.
- incrementally via  $k$  iterations. At each of the iterations  $j = 1, 2, \dots, k$ , the algorithm computes a partial BWT string  $\text{bwt}_j(S)$  by inserting the symbols preceding the  $j$ -suffixes of  $S$  at their correct positions into  $\text{bwt}_{j-1}(S)$ . Each iteration  $j$  simulates the insertion of the  $j$ -suffixes in the suffix array.
- The string  $\text{bwt}_j(S)$  is a 'partial BWT' in the sense that the addition of  $m$  end markers in their correct positions would make it the BWT of the collection  $\{S_1[k-j-1, k], S_2[k-j-1, k], \dots, S_m[k-j-1, k]\}$ .
- *This insertion does not affect the relative ordering of symbols inserted during previous iterations.*

# Observations

Let  $S$  be a collection of  $m$  strings of length  $k$  on an alphabet of  $\sigma$  letters. Our algorithm computes the BWT of  $S$

- without concatenating the strings belonging to  $S$  and without needing to compute their suffix array.
- incrementally via  $k$  iterations. At each of the iterations  $j = 1, 2, \dots, k$ , the algorithm computes a partial BWT string  $\text{bwt}_j(S)$  by inserting the symbols preceding the  $j$ -suffixes of  $S$  at their correct positions into  $\text{bwt}_{j-1}(S)$ . Each iteration  $j$  simulates the insertion of the  $j$ -suffixes in the suffix array.
- The string  $\text{bwt}_j(S)$  is a 'partial BWT' in the sense that the addition of  $m$  end markers in their correct positions would make it the BWT of the collection  $\{S_1[k-j-1, k], S_2[k-j-1, k], \dots, S_m[k-j-1, k]\}$ .
- *This insertion does not affect the relative ordering of symbols inserted during previous iterations.*



# Observations

Let  $S$  be a collection of  $m$  strings of length  $k$  on an alphabet of  $\sigma$  letters. Our algorithm computes the BWT of  $S$

- without concatenating the strings belonging to  $S$  and without needing to compute their suffix array.
- incrementally via  $k$  iterations. At each of the iterations  $j = 1, 2, \dots, k$ , the algorithm computes a partial BWT string  $\text{bwt}_j(S)$  by inserting the symbols preceding the  $j$ -suffixes of  $S$  at their correct positions into  $\text{bwt}_{j-1}(S)$ . Each iteration  $j$  simulates the insertion of the  $j$ -suffixes in the suffix array.
- The string  $\text{bwt}_j(S)$  is a ‘partial BWT’ in the sense that the addition of  $m$  end markers in their correct positions would make it the BWT of the collection  $\{S_1[k-j-1, k], S_2[k-j-1, k], \dots, S_m[k-j-1, k]\}$ .
- *This insertion does not affect the relative ordering of symbols inserted during previous iterations.*

# Observations

Let  $S$  be a collection of  $m$  strings of length  $k$  on an alphabet of  $\sigma$  letters. Our algorithm computes the BWT of  $S$

- without concatenating the strings belonging to  $S$  and without needing to compute their suffix array.
- incrementally via  $k$  iterations. At each of the iterations  $j = 1, 2, \dots, k$ , the algorithm computes a partial BWT string  $\text{bwt}_j(S)$  by inserting the symbols preceding the  $j$ -suffixes of  $S$  at their correct positions into  $\text{bwt}_{j-1}(S)$ . Each iteration  $j$  simulates the insertion of the  $j$ -suffixes in the suffix array.
- The string  $\text{bwt}_j(S)$  is a 'partial BWT' in the sense that the addition of  $m$  end markers in their correct positions would make it the BWT of the collection  $\{S_1[k-j-1, k], S_2[k-j-1, k], \dots, S_m[k-j-1, k]\}$ .
- *This insertion does not affect the relative ordering of symbols inserted during previous iterations.*

## 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 suppose that  $\$1 < \$2 < \$3 < A < C < G < T$ .

$j$ -suffix of  $S_i$  is the last  $j$  non-\$ symbols of that string and 0-suffix of  $S_i$  is  $\$i$ .

At stage  $j$ , insert the characters associated with the  $j$ -suffixes into the partial BWT.

## 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 suppose that  $\$1 < \$2 < \$3 < A < C < G < T$ .

$j$ -suffix of  $S_i$  is the last  $j$  non- $\$$  symbols of that string and 0-suffix of  $S_i$  is  $\$i$ .

At stage  $j$ , insert the characters associated with the  $j$ -suffixes into the partial BWT.

## 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
$S_1$						<i>A</i>	<i>C</i>	$\$1$
$S_2$						<i>T</i>	<i>C</i>	$\$2$
$S_3$						<i>T</i>	<i>T</i>	$\$3$

We suppose that  $\$1 < \$2 < \$3 < A < C < G < T$ .

$j$ -suffix of  $S_i$  is the last  $j$  non-\$ symbols of that string and 0-suffix of  $S_i$  is  $\$i$ .

At stage  $j$ , insert the characters associated with the  $j$ -suffixes into the partial BWT.

## 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 suppose that  $\$1 < \$2 < \$3 < A < C < G < T$ .

$j$ -suffix of  $S_i$  is the last  $j$  non-\$ symbols of that string and 0-suffix of  $S_i$  is  $\$i$ .

At stage  $j$ , insert the characters associated with the  $j$ -suffixes into the partial BWT.

## 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 suppose that  $\$1 < \$2 < \$3 < A < C < G < T$ .

$j$ -suffix of  $S_i$  is the last  $j$  non- $\$$  symbols of that string and 0-suffix of  $S_i$  is  $\$i$ .

At stage  $j$ , insert the characters associated with the  $j$ -suffixes into the partial BWT.

## 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
$S_1$			<i>C</i>	<i>C</i>	<i>A</i>	<i>A</i>	<i>C</i>	$\$1$
$S_2$			<i>A</i>	<i>G</i>	<i>C</i>	<i>T</i>	<i>C</i>	$\$2$
$S_3$			<i>C</i>	<i>G</i>	<i>C</i>	<i>T</i>	<i>T</i>	$\$3$

We suppose that  $\$1 < \$2 < \$3 < A < C < G < T$ .

$j$ -suffix of  $S_i$  is the last  $j$  non- $\$$  symbols of that string and 0-suffix of  $S_i$  is  $\$i$ .

At stage  $j$ , insert the characters associated with the  $j$ -suffixes into the partial BWT.



## 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
$S_1$		<i>G</i>	<i>C</i>	<i>C</i>	<i>A</i>	<i>A</i>	<i>C</i>	$\$1$
$S_2$		<i>G</i>	<i>A</i>	<i>G</i>	<i>C</i>	<i>T</i>	<i>C</i>	$\$2$
$S_3$		<i>T</i>	<i>C</i>	<i>G</i>	<i>C</i>	<i>T</i>	<i>T</i>	$\$3$

We suppose that  $\$1 < \$2 < \$3 < A < C < G < T$ .

$j$ -suffix of  $S_i$  is the last  $j$  non- $\$$  symbols of that string and 0-suffix of  $S_i$  is  $\$i$ .

At stage  $j$ , insert the characters associated with the  $j$ -suffixes into the partial BWT.

## 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
$S_1$	<i>T</i>	<i>G</i>	<i>C</i>	<i>C</i>	<i>A</i>	<i>A</i>	<i>C</i>	$\$1$
$S_2$	<i>A</i>	<i>G</i>	<i>A</i>	<i>G</i>	<i>C</i>	<i>T</i>	<i>C</i>	$\$2$
$S_3$	<i>G</i>	<i>T</i>	<i>C</i>	<i>G</i>	<i>C</i>	<i>T</i>	<i>T</i>	$\$3$

We suppose that  $\$1 < \$2 < \$3 < A < C < G < T$ .

$j$ -suffix of  $S_i$  is the last  $j$  non- $\$$  symbols of that string and 0-suffix of  $S_i$  is  $\$i$ .

At stage  $j$ , insert the characters associated with the  $j$ -suffixes into the partial BWT.

## 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 suppose that  $\$1 < \$2 < \$3 < A < C < G < T$ .

$j$ -suffix of  $S_i$  is the last  $j$  non- $\$$  symbols of that string and 0-suffix of  $S_i$  is  $\$i$ .

At stage  $j$ , insert the characters associated with the  $j$ -suffixes into the partial BWT.

# Iteration 0

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 obtain:

$C$
$C$
$T$

# Iteration 0

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
$S_1$							<i>C</i>	$\$1$
$S_2$							<i>C</i>	$\$2$
$S_3$							<i>T</i>	$\$3$

We obtain:

<i>C</i>
<i>C</i>
<i>T</i>

# Iteration 0

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
$S_1$							<i>C</i>	$\$1$
$S_2$							<i>C</i>	$\$2$
$S_3$							<i>T</i>	$\$3$

We obtain:

<i>C</i>
<i>C</i>
<i>T</i>

# Iteration 0

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
$S_1$						<i>A</i>	<i>C</i>	$\$1$
$S_2$						<i>T</i>	<i>C</i>	$\$2$
$S_3$						<i>T</i>	<i>T</i>	$\$3$

We obtain:

<i>C</i>
<i>C</i>
<i>T</i>

# Iteration 0

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
$S_1$						<i>A</i>	<i>C</i>	$\$1$
$S_2$						<i>T</i>	<i>C</i>	$\$2$
$S_3$						<i>T</i>	<i>T</i>	$\$3$

We obtain:

<i>C</i>
<i>C</i>
<i>T</i>



# Iteration 0

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
$S_1$						<i>A</i>	<i>C</i>	$\$1$
$S_2$						<i>T</i>	<i>C</i>	$\$2$
$S_3$						<i>T</i>	<i>T</i>	$\$3$

We obtain:

<i>C</i>
<i>C</i>
<i>T</i>

# Iteration 0

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
$S_1$						<i>A</i>	<i>C</i>	$\$1$
$S_2$						<i>T</i>	<i>C</i>	$\$2$
$S_3$						<i>T</i>	<i>T</i>	$\$3$

We obtain:

<i>C</i>
<i>C</i>
<i>T</i>

# Iteration 0

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
$S_1$						<i>A</i>	<i>C</i>	$\$1$
$S_2$						<i>T</i>	<i>C</i>	$\$2$
$S_3$						<i>T</i>	<i>T</i>	$\$3$

We obtain:

<i>C</i>
<i>C</i>
<i>T</i>

# Iteration 0

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
$S_1$						<i>A</i>	<i>C</i>	$\$1$
$S_2$						<i>T</i>	<i>C</i>	$\$2$
$S_3$						<i>T</i>	<i>T</i>	$\$3$

We obtain:

<i>C</i>
<i>C</i>
<i>T</i>

## Observation

$$LF[i] = C[L[i]] + \text{rank}(L[i], i - 1)$$

We can think of  $\text{bwt}_j(S)$  as being partitioned into  $\sigma + 1$  strings  $B_j(0), B_j(1), \dots, B_j(\sigma)$ , with the symbols in  $B_j(h)$  being those that are associated with the suffixes of  $S$  that are of length  $j$  or less and begin with  $c_0 = \$$  and  $c_h \in \Sigma$ , for  $h = 1, \dots, \sigma$ .

		$F$						$L$
		$\downarrow$						$\downarrow$
$B(0)$	0	<b>\$</b>	<i>a</i>	<i>b</i>	<i>r</i>	<i>a</i>	<i>c</i>	<i>a</i>
$B(1)$	1	<b>a</b>	<b>\$</b>	<i>a</i>	<i>b</i>	<i>r</i>	<i>a</i>	<i>c</i>
	2	<b>a</b>	<b>b</b>	<b>r</b>	<b>a</b>	<b>c</b>	<b>a</b>	<b>\$</b>
	3	<b>a</b>	<b>c</b>	<b>a</b>	<b>\$</b>	<i>a</i>	<i>b</i>	<i>r</i>
$B(2)$	4	<b>b</b>	<b>r</b>	<b>a</b>	<b>c</b>	<b>a</b>	<b>\$</b>	<i>a</i>
$B(3)$	5	<b>c</b>	<b>a</b>	<b>\$</b>	<i>a</i>	<i>b</i>	<i>r</i>	<i>a</i>
$B(4)$	6	<b>r</b>	<b>a</b>	<b>c</b>	<b>a</b>	<b>\$</b>	<i>a</i>	<i>b</i>

We do not need the array  $C$ . We only need the rank function.

We note that  $B_j(0)$  is constant for all  $j$  and, at each iteration  $j$ , we store  $B_j(h)$  in  $\sigma + 1$  external files that are sequentially read one-by-one.

## Observation

$$LF[i] = C[L[i]] + \text{rank}(L[i], i - 1)$$

We can think of  $\text{bwt}_j(S)$  as being partitioned into  $\sigma + 1$  strings  $B_j(0), B_j(1), \dots, B_j(\sigma)$ , with the symbols in  $B_j(h)$  being those that are associated with the suffixes of  $S$  that are of length  $j$  or less and begin with  $c_0 = \$$  and  $c_h \in \Sigma$ , for  $h = 1, \dots, \sigma$ .

		$F$						$L$
		↓						↓
$B(0)$	0	<b>\$</b>	<i>a</i>	<i>b</i>	<i>r</i>	<i>a</i>	<i>c</i>	<i>a</i>
$B(1)$	1	<b>a</b>	<b>\$</b>	<i>a</i>	<i>b</i>	<i>r</i>	<i>a</i>	<i>c</i>
	2	<b>a</b>	<b>b</b>	<b>r</b>	<b>a</b>	<b>c</b>	<b>a</b>	<b>\$</b>
	3	<b>a</b>	<b>c</b>	<b>a</b>	<b>\$</b>	<i>a</i>	<i>b</i>	<i>r</i>
$B(2)$	4	<b>b</b>	<b>r</b>	<b>a</b>	<b>c</b>	<b>a</b>	<b>\$</b>	<i>a</i>
$B(3)$	5	<b>c</b>	<b>a</b>	<b>\$</b>	<i>a</i>	<i>b</i>	<i>r</i>	<i>a</i>
$B(4)$	6	<b>r</b>	<b>a</b>	<b>c</b>	<b>a</b>	<b>\$</b>	<i>a</i>	<i>b</i>

We do not need the array  $C$ . We only need the rank function.

We note that  $B_j(0)$  is constant for all  $j$  and, at each iteration  $j$ , we store  $B_j(h)$  in  $\sigma + 1$  external files that are sequentially read one-by-one.

## Observation

$$LF[i] = C[L[i]] + \text{rank}(L[i], i - 1)$$

We can think of  $\text{bwt}_j(S)$  as being partitioned into  $\sigma + 1$  strings  $B_j(0), B_j(1), \dots, B_j(\sigma)$ , with the symbols in  $B_j(h)$  being those that are associated with the suffixes of  $S$  that are of length  $j$  or less and begin with  $c_0 = \$$  and  $c_h \in \Sigma$ , for  $h = 1, \dots, \sigma$ .

		$F$						$L$
		$\downarrow$						$\downarrow$
$B(0)$	0	<b>\$</b>	<i>a</i>	<i>b</i>	<i>r</i>	<i>a</i>	<i>c</i>	<i>a</i>
$B(1)$	1	<b>a</b>	<b>\$</b>	<i>a</i>	<i>b</i>	<i>r</i>	<i>a</i>	<i>c</i>
	2	<b>a</b>	<b>b</b>	<b>r</b>	<b>a</b>	<b>c</b>	<b>a</b>	<b>\$</b>
	3	<b>a</b>	<b>c</b>	<b>a</b>	<b>\$</b>	<i>a</i>	<i>b</i>	<i>r</i>
$B(2)$	4	<b>b</b>	<b>r</b>	<b>a</b>	<b>c</b>	<b>a</b>	<b>\$</b>	<i>a</i>
$B(3)$	5	<b>c</b>	<b>a</b>	<b>\$</b>	<i>a</i>	<i>b</i>	<i>r</i>	<i>a</i>
$B(4)$	6	<b>r</b>	<b>a</b>	<b>c</b>	<b>a</b>	<b>\$</b>	<i>a</i>	<i>b</i>

We do not need the array  $C$ . We only need the rank function.

We note that  $B_j(0)$  is constant for all  $j$  and, at each iteration  $j$ , we store  $B_j(h)$  in  $\sigma + 1$  external files that are sequentially read one-by-one.

## Observation

$$LF[i] = C[L[i]] + \text{rank}(L[i], i - 1)$$

We can think of  $\text{bwt}_j(S)$  as being partitioned into  $\sigma + 1$  strings  $B_j(0), B_j(1), \dots, B_j(\sigma)$ , with the symbols in  $B_j(h)$  being those that are associated with the suffixes of  $S$  that are of length  $j$  or less and begin with  $c_0 = \$$  and  $c_h \in \Sigma$ , for  $h = 1, \dots, \sigma$ .

		$F$						$L$
		↓						↓
$B(0)$	0	<b>\$</b>	<i>a</i>	<i>b</i>	<i>r</i>	<i>a</i>	<i>c</i>	<i>a</i>
$B(1)$	1	<b>a</b>	<b>\$</b>	<i>a</i>	<i>b</i>	<i>r</i>	<i>a</i>	<i>c</i>
	2	<b>a</b>	<b>b</b>	<b>r</b>	<b>a</b>	<b>c</b>	<b>a</b>	<b>\$</b>
	3	<b>a</b>	<b>c</b>	<b>a</b>	<b>\$</b>	<i>a</i>	<i>b</i>	<i>r</i>
$B(2)$	4	<b>b</b>	<b>r</b>	<b>a</b>	<b>c</b>	<b>a</b>	<b>\$</b>	<i>a</i>
$B(3)$	5	<b>c</b>	<b>a</b>	<b>\$</b>	<i>a</i>	<i>b</i>	<i>r</i>	<i>a</i>
$B(4)$	6	<b>r</b>	<b>a</b>	<b>c</b>	<b>a</b>	<b>\$</b>	<i>a</i>	<i>b</i>

We do not need the array  $C$ . We only need the rank function.

We note that  $B_j(0)$  is constant for all  $j$  and, at each iteration  $j$ , we store  $B_j(h)$  in  $\sigma + 1$  external files that are sequentially read one-by-one.



## Looking in detail at iteration 6

$B_5(0)$	Associated Suffixes	$T$ <b>G</b> CCAAC\$ <sub>1</sub> ,	$B_6(0)$	Associated Suffixes	
0	C	A <b>G</b> AGCTC\$ <sub>2</sub> ,	0	C	\$ <sub>1</sub>
1	C	G <b>T</b> CGCTT\$ <sub>3</sub> .	1	C	\$ <sub>2</sub>
2	T		2	T	\$ <sub>3</sub>
		$P_5(0) = [], N_5(0) = []$ (empty array)			
$B_5(1)$	Associated Suffixes	$P_5(1) = [2], N_5(1) = [2]$	$B_6(1)$	Associated Suffixes	
0	C	$P_5(2) = [3, 4], N_5(2) = [1, 3]$	0	C	AAC\$ <sub>1</sub>
1	A	$P_5(3) = [], N_5(3) = []$	1	A	AC\$ <sub>1</sub>
2	<b>G</b>	$P_5(4) = [], N_5(4) = []$	2	G	AGCTC\$ <sub>2</sub>
		↓			
$B_5(2)$	Associated Suffixes	For $h = 0, 3, 4$ : nothing	$B_6(2)$	Associated Suffixes	
0	A	For $h = 1$ :	0	A	C\$ <sub>1</sub>
1	T	$rank(G, 2) = 0$ (sequence = 2)	1	T	C\$ <sub>2</sub>
2	C	For $h = 2$ :	2	C	CAAC\$ <sub>1</sub>
3	<b>G</b>	$rank(G, 3) = 1$ (sequence = 1)	3	G	CCAAC\$ <sub>1</sub>
4	<b>T</b>	$rank(T, 4) = 2$ (sequence = 3)	4	T	CGCTT\$ <sub>3</sub>
5	G		5	G	CTC\$ <sub>2</sub>
6	G		6	G	CTT\$ <sub>3</sub>
		↓			
$B_5(3)$	Associated Suffixes	$T$ <b>G</b> CCAAC\$ <sub>1</sub> ,	$B_6(3)$	Associated Suffixes	
0	A	A <b>G</b> AGCTC\$ <sub>2</sub> ,	0	<b>A</b>	<b>G</b> AGCTC\$ <sub>2</sub>
1	C	G <b>T</b> CGCTT\$ <sub>3</sub> .	1	<b>T</b>	<b>G</b> CCAAC\$ <sub>1</sub>
		↓	2	A	GCTC\$ <sub>2</sub>
$B_5(4)$	Associated Suffixes	$P_6(0) = [], N_6(0) = []$	3	C	GCTT\$ <sub>3</sub>
0	T	$P_6(1) = [], N_6(1) = []$			
1	C	$P_6(2) = [], N_6(2) = []$	$B_6(4)$	Associated Suffixes	
2	C	$P_6(3) = [0, 1]$ and $N_6(3) = [2, 1]$	0	T	T\$ <sub>3</sub>
		$P_6(4) = [2]$ and $N_6(4) = [3]$	1	C	TC\$ <sub>2</sub>
			2	<b>G</b>	<b>T</b> CGCTT\$ <sub>3</sub>
			3	C	TT\$ <sub>3</sub>

Position of  $G$ CCAAC\$<sub>1</sub> in  $G$  segment = # of  $G$  before  $CCAAC$ \$<sub>1</sub> in partial BWT = # of  $G$  in \$-segment + # of  $G$  in

A-segment + # of  $G$  before  $CCAAC$ \$<sub>1</sub> in  $C$ -segment

## Looking in detail at iteration 6

$B_5(0)$	Associated Suffixes	$T$ <b>G</b> CCAAC\$ <sub>1</sub> ,	$B_6(0)$	Associated Suffixes
0	C	A <b>G</b> AGCTC\$ <sub>2</sub> ,	0	C
1	C	G <b>T</b> CGCTT\$ <sub>3</sub> .	1	C
2	T		2	T
		$P_5(0) = [], N_5(0) = []$ (empty array)		
$B_5(1)$	Associated Suffixes	$P_5(1) = [2], N_5(1) = [2]$	$B_6(1)$	Associated Suffixes
0	C	$P_5(2) = [3, 4], N_5(2) = [1, 3]$	0	C
1	A	$P_5(3) = [], N_5(3) = []$	1	A
2	<b>G</b>	$P_5(4) = [], N_5(4) = []$	2	G
		↓		
$B_5(2)$	Associated Suffixes	For $h = 0, 3, 4$ : nothing	$B_6(2)$	Associated Suffixes
0	A	For $h = 1$ :	0	A
1	T	$rank(G, 2) = 0$ (sequence = 2)	1	T
2	C	For $h = 2$ :	2	C
3	<b>G</b>	$rank(G, 3) = 1$ (sequence = 1)	3	G
4	<b>T</b>	$rank(T, 4) = 2$ (sequence = 3)	4	T
5	G		5	G
6	G		6	G
		↓		
$B_5(3)$	Associated Suffixes	$T$ GCCAAC\$ <sub>1</sub> ,	$B_6(3)$	Associated Suffixes
0	A	A <b>G</b> AGCTC\$ <sub>2</sub> ,	0	<b>A</b>
1	C	G <b>T</b> CGCTT\$ <sub>3</sub> .	1	<b>T</b>
		↓	2	A
$B_5(4)$	Associated Suffixes	$P_6(0) = [], N_6(0) = []$	3	C
0	T	$P_6(1) = [], N_6(1) = []$		
1	C	$P_6(2) = [], N_6(2) = []$	$B_6(4)$	Associated Suffixes
2	C	$P_6(3) = [0, 1]$ and $N_6(3) = [2, 1]$	0	T
		$P_6(4) = [2]$ and $N_6(4) = [3]$	1	C
			2	<b>G</b>
			3	C

Position of  $GCCAAC$$ <sub>1</sub> in  $G$  segment = # of  $G$  before  $CCAAC$$ <sub>1</sub> in partial BWT = # of  $G$  in  $\$$ -segment + # of  $G$  in

$A$ -segment + # of  $G$  before  $CCAAC$$ <sub>1</sub> in  $C$ -segment

## Two versions of our algorithm: BCR vs. BCRext

	<b>BCR</b>	<b>BCRext</b>
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)	$O(mk^2\log(s))$ (partial BWT) $O(mk\log(\sigma))$ (sequence slices)	$O(mk^2\log(\sigma))$ (partial BWT) $O(mk^2\log(\sigma))$ ( <i>sequences</i> ) $O(mk\log(mk))$ ( <i>P - array</i> ) $O(mk\log(m))$ ( <i>N - array</i> )

# Performance on human DNA sequence data

<i>Dataset size (millions of 100-mers)</i>	<i>Program Program</i>	<i>Wallclock time (<math>\mu</math>s per input base)</i>	<i>CPU efficiency (%)</i>	<i>Max RAM (Gbyte)</i>
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

# Further works

- Able to compute BWT of 1 billion 100-mers in under 24 hours
- Ongoing work:
  - Further optimizations to construction, parallelization
  - Software library for construction/querying of BWT of large string collections
  - Algorithm can be adapted to allow sets of strings to be added/removed from collection
  - Applications of BWT of string collection to bioinformatics