Funded by
the European Union
NextGenerationEU

A class of heuristics for reducing the number of BWT-runs in the String Ordering Problem

G. Bertola[1], A. J. Cox[2], V. Guerrini[1], G. Rosone[1]

[1]Department of Computer Science, University of Pisa, Italy
[2]Genomerics Ltd, Cambridge, UK

Combinatorial Pattern Matching 2024, June 25, 2024

Work partially supported by PNRR THE (Tuscany Health Ecosystem) - Spoke 6 "Precision medicine & personalized healthcare" - funded by the EU under the NextGeneration EU programme.

The Burrows-Wheeler Transform (BWT)

The Burrows-Wheeler Transform is a **reversible** text transformation that given a string S over an ordered alphabet Σ outputs:

- BWT(S): a permutation of the characters of S
- I: the position of S in the sorted list of rotations of S

Ex. $S = mathematics$ (11 runs)

BWT(S)$= mmihttsecaa$, $I = 7$ (8 runs)

Clustering effect: equal symbols followed by a similar context are grouped

BWT tends to reduce the number of **runs** of a same symbol

The Burrows-Wheeler Transform (BWT)

The Burrows-Wheeler Transform is a **reversible** text transformation that given a string S over an ordered alphabet Σ outputs:

- BWT(S): a permutation of the characters of S
- I: the position of S in the sorted list of rotations of S

Ex. $S = mathematics$ (11 runs)

BWT(S)$= mmihttsecaa$, $I = 7$ (8 runs)

Clustering effect: equal symbols followed by a similar context are grouped

The number of equal-symbol runs is a relevant BWT parameter

The extended Burrows-Wheeler Transform

Abundance of string collections $\rightarrow$ focus shifted from a single string to a collection

1. definition via a new order relation (called ω-order) among the cyclic rotations of the input strings [Mantaci, Restivo, Rosone and Sciortino, 2007]
2. definition by appending end-markers to each string and sorting suffixes of the input strings [Bauer, Cox and Rosone, 2013]

The extended Burrows-Wheeler Transform

Abundance of string collections → focus shifted from a single string to a collection

1. definition via a new order relation (called ω-order) among the cyclic rotations of the input strings [Mantaci, Restivo, Rosone and Sciortino, 2007]
2. definition by appending end-markers to each string and sorting suffixes of the input strings [Bauer, Cox and Rosone, 2013]

Properties:

- reversible transformation producing a permutation of the symbols of the input string collection
- clustering effect (generally reduces the number of runs)
- reconstruction of any subset of the input collection
- strings can be added/removed (dynamic BWT)
- the strings are not concatenated

The extended Burrows-Wheeler Transform [Bauer et al., 2013]

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

1. Append to S_i an end-marker $\$_i$, where $\$_i < a$, for any $a \in \Sigma$, and $\$_i < \$_j$, if $i < j$.
2. Sort all the suffixes of the strings in $\mathcal{S}'$ lexicographically.
3. Output the string obtained by concatenating the symbols that (circularly) precede each first symbol of the suffixes in the list.

BWT($\mathcal{S}'$)=ATTTAGTGCCTG$\$_3\$_1CCC\$_2T\$_4$

$\mathcal{S}' = \{\text{GGAA}\$_1, \text{TCCT}\$_2, \text{GCCT}\$_3, \text{TTCT}\$_4\}$

BWT	Sorted suffixes
A	$\$_1$
T	$\$_2$
T	$\$_3$
T	$\$_4$
A	A$\$_1$
G	AA$\$_1$
T	CCT$\$_2$
G	CCT$\$_3$
C	CT$\$_2$
C	CT$\$_3$
T	CT$\$_4$
G	GAA$\$_1$
$\$_3$	GCCT$\$_3$
$\$_1$	GGAA$\$_1$
C	T$\$_2$
C	T$\$_3$
C	T$\$_4$
$\$_2$	TCCT$\$_2$
T	TCT$\$_4$
$\$_4$	TTCT$\$_4$

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

1. Append to S_i an end-marker $\$_i$, where $\$_i < a$, for any $a \in \Sigma$, and $\$_i < \$_j$, if $i < j$.
2. Sort all the suffixes of the strings in $\mathcal{S}'$ lexicographically.
3. Output the string obtained by concatenating the symbols that (circularly) precede each first symbol of the suffixes in the list.

$\mathcal{S}' = \{GGAA\$_1, TCCT\$_2, GCCT\$_3, TTCT\$_4\}$

BWT	Sorted suffixes
A	$\$_1$
T	$\$_2$
T	$\$_3$
T	$\$_4$
A	A$\$_1$
G	AA$\$_1$
T	CCT$\$_2$
G	CCT$\$_3$
C	CT$\$_2$
C	CT$\$_3$
T	CT$\$_4$
G	GAA$\$_1$
$\$_3$	GCCT$\$_3$
$\$_1$	GGAA$\$_1$
C	T$\$_2$
C	T$\$_3$
C	T$\$_4$
$\$_2$	TCCT$\$_2$
T	TCT$\$_4$
$\$_4$	TTCT$\$_4$

The extended Burrows-Wheeler Transform [Bauer et al., 2013]

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

1. Append to S_i an end-marker $\$_i$, where $\$_i < a$, for any $a \in \Sigma$, and $\$_i < \$_j$, if $i < j$.
2. Sort all the suffixes of the strings in $\mathcal{S}'$ lexicographically.
3. Output the string obtained by concatenating the symbols that (circularly) precede each first symbol of the suffixes in the list.

Using different and ordered end-markers gives an order on the string collection

$\mathcal{S}'' =$ {GGAA$\$_1$, TCCT$\$_2$, TTCT$\$_3$, GCCT$\$_4$}

BWT	Sorted suffixes
A	$\$_1$
T	$\$_2$
T	$\$_3$
T	$\$_3$
A	A$\$_1$
G	AA$\$_1$
T	CCT$\$_2$
G	CCT$\$_4$
C	CT$\$_2$
T	CT$\$_3$
C	CT$\$_4$
G	GAA$\$_1$
$\$_4$	GCCT$\$_4$
$\$_1$	GGAA$\$_1$
C	T$\$_2$
C	T$\$_3$
C	T$\$_4$
$\$_2$	TCCT$\$_2$
T	TCT$\$_3$
$\$_3$	TTCT$\$_3$

Same-As-Previous intervals

SAP interval [Cox et al., 2012]:
maximal segment BWT$[i, j]$
such that any suffix in $[i + 1, j]$
is same-as-previous
(up to the end-marker)

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

SAP interval (rows A, C, A, A, C)

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

Remark 1. The BWT strings of $\mathcal{S}'$ and $\mathcal{S}''$ can only differ within SAP-intervals that contain more than one distinct symbol

Remark 2. Within a SAP-interval, the reordering of the characters implicitly involves permuting the strings in the collection

See also [Bentley et al., ESA 2020] and [Cenzato and Lipták, Bioinformatics 2024]

A class of heuristics for reducing the BWT-runs

Definition.

Given a string collection $\mathcal{S}$, the class of transformed strings $\mathfrak{S}_{\mathcal{S}}$ comprises the BWT strings obtained by sorting symbols in some SAP-intervals according to a different ***adaptive*** alphabet ordering

Previous heuristics and **optBWT**:

- Cox et al. (2012) experimentally showed a reduced number of runs in the BWT string by permuting symbols in SAP-intervals. **Two heuristics**: **rloBWT** and **sapBWT**
- Bentley et al. (2020) designed a linear-time algorithm for reordering symbols in the BWT so that it yields the **minimum number of runs**
  Cenzato et al. (2023) used such a ***post-processing* strategy** to actually compute the optimal BWT

A class of heuristics for reducing the BWT-runs

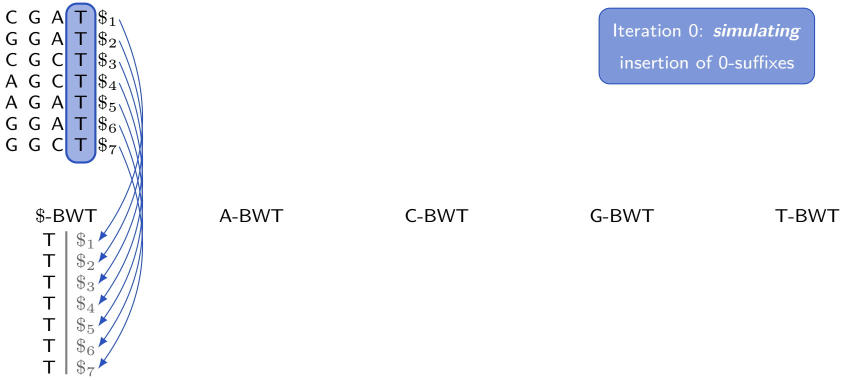
Definition.

Given a string collection $\mathcal{S}$, the class of transformed strings $\mathfrak{S}_{\mathcal{S}}$ comprises the BWT strings obtained by sorting symbols in some SAP-intervals according to a different ***adaptive*** alphabet ordering

- **New heuristics** for reducing the number of runs **while computing the transformed string**
- New heuristics **improve on** the number of runs of **previously-introduced heuristics** by Cox et al.
- Output string obtained assuming **no *a priori* ordering** of the end-markers
- Symbols are sorted during an incremental construction that parses suffixes of the same length, like BCR algorithm does

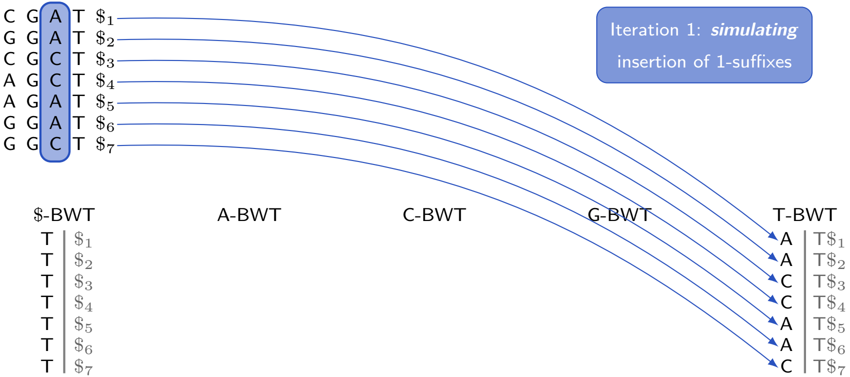
BCR Algorithm [Bauer et al., 2013]

- BCR algorithm builds the BWT of a collection of strings *incrementally* by right-to-left scanning all the strings simultaneously



BCR Algorithm [Bauer et al., 2013]

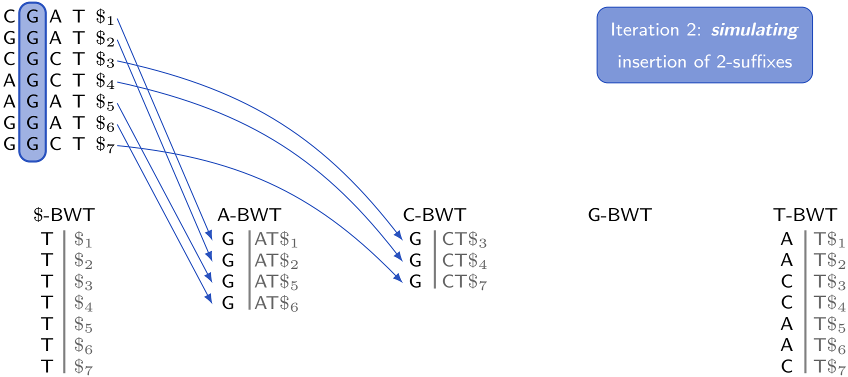
- BCR algorithm builds the BWT of a collection of strings *incrementally* by right-to-left scanning all the strings simultaneously



BCR Algorithm [Bauer et al., 2013]

- BCR algorithm builds the BWT of a collection of strings *incrementally* by right-to-left scanning all the strings simultaneously

Iteration 2: ***simulating*** insertion of 2-suffixes



BCR Algorithm [Bauer et al., 2013]

- BCR algorithm builds the BWT of a collection of strings *incrementally* by right-to-left scanning all the strings simultaneously

C G A T $\$_1$
G G A T $\$_2$
C G C T $\$_3$
A G C T $\$_4$
A G A T $\$_5$
G G A T $\$_6$
G G C T $\$_7$

Last iteration:
insertion of end-markers

$-BWT

T	$\$_1$
T	$\$_2$
T	$\$_3$
T	$\$_4$
T	$\$_5$
T	$\$_6$
T	$\$_7$

A-BWT

$	AGAT$\$_5$
$	AGCT$\$_4$
G	AT$\$_1$
G	AT$\$_2$
G	AT$\$_5$
G	AT$\$_6$

C-BWT

$	CGAT$\$_1$
$	CGCT$\$_3$
G	CT$\$_3$
G	CT$\$_4$
G	CT$\$_7$

G-BWT

$	GGAT$\$_2$
$	GGAT$\$_6$
$	GGCT$\$_7$
C	GAT$\$_1$
G	GAT$\$_2$
A	GAT$\$_5$
G	GAT$\$_6$
C	GCT$\$_3$
A	GCT$\$_4$
G	GCT$\$_7$

T-BWT

A	T$\$_1$
A	T$\$_2$
C	T$\$_3$
C	T$\$_4$
A	T$\$_5$
A	T$\$_6$
C	T$\$_7$

BCR Algorithm and SAP status

- BCR algorithm builds the BWT of a collection of strings *incrementally* by right-to-left scanning all the strings simultaneously
- Symbols in any SAP-interval can be first permuted, then inserted

C	G	A	T	$\$_1$
G	G	A	T	$\$_2$
C	G	C	T	$\$_3$
A	G	C	T	$\$_4$
A	G	A	T	$\$_5$
G	G	A	T	$\$_6$
G	G	C	T	$\$_7$

$-BWT		A-BWT	C-BWT	G-BWT	T-BWT	
T	$					T$
T	$					T$
T	$					T$
T	$					T$
T	$					T$
T	$					T$
T	$					T$

BCR Algorithm and SAP status

- BCR algorithm builds the BWT of a collection of strings *incrementally* by right-to-left scanning all the strings simultaneously
- Symbols in any SAP-interval can be first permuted, then inserted

C	G	A	T	$\$_1$
G	G	A	T	$\$_2$
C	G	C	T	$\$_3$
A	G	C	T	$\$_4$
A	G	A	T	$\$_5$
G	G	A	T	$\$_6$
G	G	C	T	$\$_7$

$-BWT

T	$
T	$
T	$
T	$
T	$
T	$
T	$

A-BWT

C-BWT

G-BWT

T-BWT

A	T$
A	T$
A	T$
A	T$
C	T$
C	T$
C	T$

BCR Algorithm and SAP status

- BCR algorithm builds the BWT of a collection of strings *incrementally* by right-to-left scanning all the strings simultaneously
- Symbols in any SAP-interval can be first permuted, then inserted

C	G	A	T	$\$_1$
G	G	A	T	$\$_2$
C	G	C	T	$\$_3$
A	G	C	T	$\$_4$
A	G	A	T	$\$_5$
G	G	A	T	$\$_6$
G	G	C	T	$\$_7$

$-BWT

T	$
T	$
T	$
T	$
T	$
T	$
T	$

A-BWT

G	AT$
G	AT$
G	AT$
G	AT$

C-BWT

G	CT$
G	CT$
G	CT$

G-BWT

A	GAT$
C	GAT$
G	GAT$
G	GAT$

T-BWT

A	T$
A	T$
A	T$
A	T$
C	T$
C	T$
C	T$

BCR Algorithm and SAP status

- BCR algorithm builds the BWT of a collection of strings *incrementally* by right-to-left scanning all the strings simultaneously
- Symbols in any SAP-interval can be first permuted, then inserted

C	G	A	T	$\$_1$
G	G	A	T	$\$_2$
C	G	C	T	$\$_3$
A	G	C	T	$\$_4$
A	G	A	T	$\$_5$
G	G	A	T	$\$_6$
G	G	C	T	$\$_7$

$-BWT

T	$
T	$
T	$
T	$
T	$
T	$
T	$

A-BWT

G	AT$
G	AT$
G	AT$
G	AT$

C-BWT

G	CT$
G	CT$
G	CT$

G-BWT

A	GAT$
C	GAT$
G	GAT$
G	GAT$
G	GCT$
C	GCT$
A	GCT$

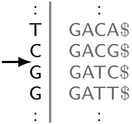
T-BWT

A	T$
A	T$
A	T$
A	T$
C	T$
C	T$
C	T$

New SAP-heuristics

altBWT alternating lexicographic order for inserting consecutive SAP-intervals

plusBWT *ad hoc* alphabet order for each SAP-interval on the basis of previously inserted symbols



New SAP-heuristics

altBWT alternating lexicographic order for inserting consecutive SAP-intervals

plusBWT *ad hoc* alphabet order for each SAP-interval on the basis of previously inserted symbols

:	:
T	GACA$
C	GACG$
A	**GATAG**$
C	**GATAG**$
A	**GATAG**$
G	**GATAG**$
G	GATC$
G	GATT$
:	:

New SAP-heuristics

altBWT alternating lexicographic order for inserting consecutive SAP-intervals

plusBWT *ad hoc* alphabet order for each SAP-interval on the basis of previously inserted symbols

⋮	⋮
T	GACA\$
C	GACG\$
C	**GATAG**\$
A	**GATAG**\$
A	**GATAG**\$
G	**GATAG**\$
G	GATC\$
G	GATT\$
⋮	⋮

New SAP-heuristics

altBWT alternating lexicographic order for inserting consecutive SAP-intervals

plusBWT *ad hoc* alphabet order for each SAP-interval on the basis of previously inserted symbols

:	:
T	GACA$
C	GACG$
C	**GATAG**$
A	**GATAG**$
A	**GATAG**$
G	**GATAG**$
G	GATC$
G	GATT$
:	:

randBWT random alphabet order for each SAP-interval

New SAP-heuristics correspond to a string reordering *not obtained a-priori*

How do SAP-heuristics perform?

- New heuristics integrated into the semi-external memory BCR implementation
- We designed some tests on real-life biological datasets

	Dataset	Description	BWT length	Max len.	No. strings	optBWT
1	*pdb_seqres*	*proteins*	241,121,574	16,181	865,773	16,829,629
2	SRR7494928–30	*Epstein Barr Virus*	984,191,064	101	9,648,932	40,700,607
3	ERR732065–70	*HIV-virus*	1,345,713,812	150	8,912,012	11,539,661
4	SRR12038540	*SARS-CoV-2 RBD*	1,690,229,250	50	33,141,750	14,864,523
5	ERR022075_1	*E. Coli str. K-12*	2,294,730,100	100	22,720,100	71,203,469
6	SRR059298	*Deformed wing virus*	2,455,299,082	72	33,634,234	48,376,632
7	SRR065389–90	*C. Elegans*	14,095,870,474	100	139,563,074	921,561,895
8	SRR2990914_1	*Sindibis virus*	15,957,722,119	36	431,289,787	105,250,120
9	ERR1019034	*H. Sapiens*	123,506,926,658	100	1,222,840,858	10,860,229,434

How do SAP-heuristics perform?

	inputBWT	Different heuristics				
		rloBWT	sapBWT	plusBWT	altBWT	randBWT
1	17,971,532	16,862,960	-	**16,848,496**	16,861,264	16,861,897
2	254,663,327	41,730,649	65,040,263	**41,372,530**	41,592,394	41,599,327
3	48,727,709	11,941,093	17,662,811	**11,766,827**	11,858,536	11,872,578
4	209,136,502	17,026,009	17,949,348	**15,226,766**	16,014,506	16,626,930
5	259,821,570	75,846,202	92,304,201	**74,529,428**	75,239,739	75,332,300
6	249,873,376	50,495,777	75,142,244	**49,619,150**	50,207,432	50,302,961
7	2,251,887,226	968,098,124	1,066,534,827	**954,489,749**	960,811,214	963,741,035
8	3,313,966,937	109,772,697	188,817,402	**108,466,351**	109,365,518	109,599,875
9	23,084,021,291	11,312,737,256	12,151,830,264	**11,179,873,104**	11,250,843,471	11,273,506,405

- plusBWT gives the fewest runs among all the heuristics, improving the number of inputBWT runs by at least half for all DNA datasets
- plusBWT, altBWT, randBWT, and rloBWT, share similar space-time performances
- On the largest dataset, computing plusBWT has $1.13\times$ time overhead and memory usage almost the same w.r.t. inputBWT

Reversibility and LF mapping

$\mathcal{S}' = \{GGAA\$_1, TCCT\$_2, GCCT\$_3, TTCT\$_4\}$

- LF Mapping: the i-th occurrence of $c \in \Sigma$ in L corresponds to the i-th occurrence of $c \in \Sigma$ in F
- For all i, the symbol $F[i]$ (circularly) follows $L[i]$ in the original (corresponding) string.

	F	L	
1	$\$_1$	A	1
2	$\$_2$	T	2
3	$\$_3$	T	3
4	$\$_4$	T	4
5	A	A	5
6	A	G	6
7	C	T	7
8	C	G	8
9	C	C	9
10	C	C	10
11	C	T	11
12	G	G	12
13	G	$\$_3$	13
14	G	$\$_1$	14
15	T	C	15
16	T	C	16
17	T	C	17
18	T	$\$_2$	18
19	T	T	19
20	T	$\$_4$	20

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

Reversibility and LF mapping

$\mathcal{S}' = \{GGAA\$_1, TCCT\$_2, GCCT\$_3, TTCT\$_4\}$

- LF Mapping: the i-th occurrence of $c \in \Sigma$ in L corresponds to the i-th occurrence of $c \in \Sigma$ in F
- For all i, the symbol $F[i]$ (circularly) follows $L[i]$ in the original (corresponding) string.

$S_2 = \quad T$

	F	L	
1	$\$_1$	A	1
2	$\$_2$	T	2
3	$\$_3$	T	3
4	$\$_4$	T	4
5	A	A	5
6	A	G	6
7	C	T	7
8	C	G	8
9	C	C	9
10	C	C	10
11	C	T	11
12	G	G	12
13	G	$\$_3$	13
14	G	$\$_1$	14
15	T	C	15
16	T	C	16
17	T	C	17
18	T	$\$_2$	18
19	T	T	19
20	T	$\$_4$	20

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

Cycle decomposition of π_{LF}: (**2** 15)

Reversibility and LF mapping

$\mathcal{S}' = \{GGAA\$_1, TCCT\$_2, GCCT\$_3, TTCT\$_4\}$

- LF Mapping:
  the i-th occurrence of $c \in \Sigma$ in L corresponds to the i-th occurrence of $c \in \Sigma$ in F

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

$S_2 = \quad CT$

	F	L	
1	$\$_1$	A	1
2	$\$_2$	T	2
3	$\$_3$	T	3
4	$\$_4$	T	4
5	A	A	5
6	A	G	6
7	C	T	7
8	C	G	8
9	C	C	9
10	C	C	10
11	C	T	11
12	G	G	12
13	G	$\$_3$	13
14	G	$\$_1$	14
15	T	C	15
16	T	C	16
17	T	C	17
18	T	$\$_2$	18
19	T	T	19
20	T	$\$_4$	20

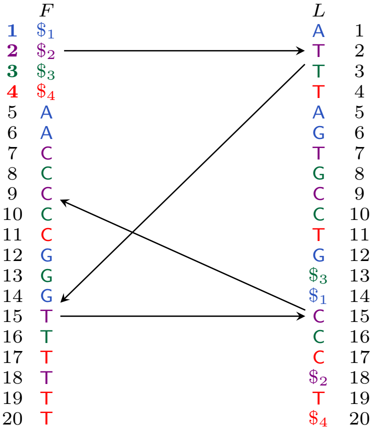
$$\pi_{LF} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 \\ 5 & 15 & 16 & 17 & 6 & 12 & 18 & 13 & 7 & 8 & 19 & 14 & \mathbf{3} & \mathbf{1} & 9 & 10 & 11 & \mathbf{2} & 20 & \mathbf{4} \end{pmatrix}$$

Cycle decomposition of π_{LF}: (**2** 15)

Reversibility and LF mapping

$\mathcal{S}' = \{GGAA\$_1, TCCT\$_2, GCCT\$_3, TTCT\$_4\}$

- LF Mapping: the i-th occurrence of $c \in \Sigma$ in L corresponds to the i-th occurrence of $c \in \Sigma$ in F
- For all i, the symbol $F[i]$ (circularly) follows $L[i]$ in the original (corresponding) string.



$S_2 = \quad CT$

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

Cycle decomposition of π_{LF}: (**2** 15 9)

Reversibility and LF mapping

$\mathcal{S}' = \{GGAA\$_1, TCCT\$_2, GCCT\$_3, TTCT\$_4\}$

- LF Mapping: the i-th occurrence of $c \in \Sigma$ in L corresponds to the i-th occurrence of $c \in \Sigma$ in F

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

$S_2 = \quad CCT$

	F	L	
1	$\$_1$	A	1
2	$\$_2$	T	2
3	$\$_3$	T	3
4	$\$_4$	T	4
5	A	A	5
6	A	G	6
7	C	T	7
8	C	G	8
9	C	C	9
10	C	C	10
11	C	T	11
12	G	G	12
13	G	$\$_3$	13
14	G	$\$_1$	14
15	T	C	15
16	T	C	16
17	T	C	17
18	T	$\$_2$	18
19	T	T	19
20	T	$\$_4$	20

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

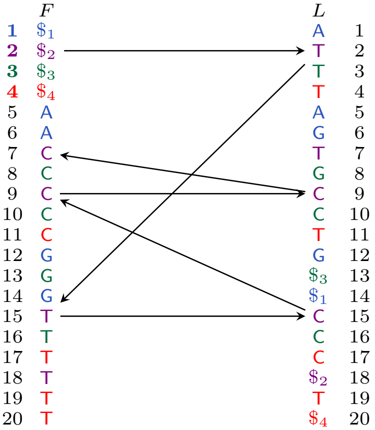
Cycle decomposition of π_{LF}: (**2** 15 9)

Reversibility and LF mapping

$\mathcal{S}' = \{GGAA\$_1, TCCT\$_2, GCCT\$_3, TTCT\$_4\}$

- LF Mapping: the i-th occurrence of $c \in \Sigma$ in L corresponds to the i-th occurrence of $c \in \Sigma$ in F
- For all i, the symbol $F[i]$ (circularly) follows $L[i]$ in the original (corresponding) string.

$S_2 = \quad CCT$



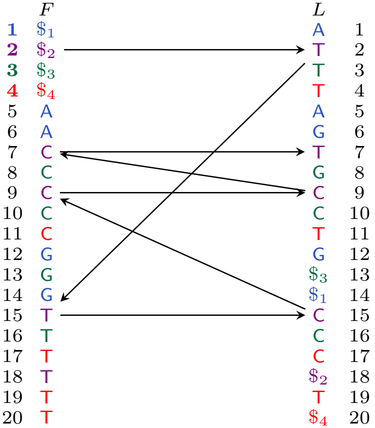
$$\pi_{LF} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 \\ 5 & 15 & 16 & 17 & 6 & 12 & 18 & 13 & 7 & 8 & 19 & 14 & \mathbf{3} & \mathbf{1} & 9 & 10 & 11 & \mathbf{2} & 20 & \mathbf{4} \end{pmatrix}$$

Cycle decomposition of π_{LF}: $(\mathbf{2}\ 15\ 9\ 7\ )$

Reversibility and LF mapping

$\mathcal{S}' = \{GGAA\$_1, TCCT\$_2, GCCT\$_3, TTCT\$_4\}$

- LF Mapping: the i-th occurrence of $c \in \Sigma$ in L corresponds to the i-th occurrence of $c \in \Sigma$ in F
- For all i, the symbol $F[i]$ (circularly) follows $L[i]$ in the original (corresponding) string.



$S_2 = \quad TCCT$

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

Cycle decomposition of π_{LF}: $(\mathbf{2}\ 15\ 9\ 7\ )$

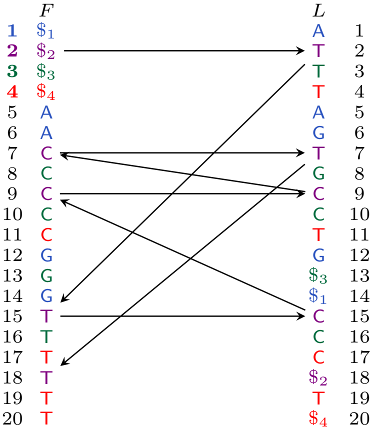
Reversibility and LF mapping

$\mathcal{S}' = \{GGAA\$_1, TCCT\$_2, GCCT\$_3, TTCT\$_4\}$

- LF Mapping: the i-th occurrence of $c \in \Sigma$ in L corresponds to the i-th occurrence of $c \in \Sigma$ in F

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

$S_2 = \quad TCCT$



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

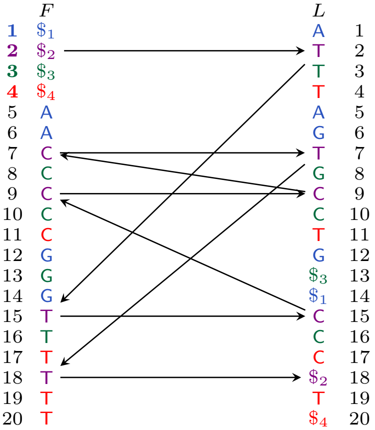
Cycle decomposition of π_{LF}: (**2** 15 9 7 18)

Reversibility and LF mapping

$\mathcal{S}' = \{GGAA\$_1, TCCT\$_2, GCCT\$_3, TTCT\$_4\}$

- LF Mapping: the i-th occurrence of $c \in \Sigma$ in L corresponds to the i-th occurrence of $c \in \Sigma$ in F
- For all i, the symbol $F[i]$ (circularly) follows $L[i]$ in the original (corresponding) string.

$S_2 = \$_2TCCT$



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

Cycle decomposition of π_{LF}: (**2** 15 9 7 18)

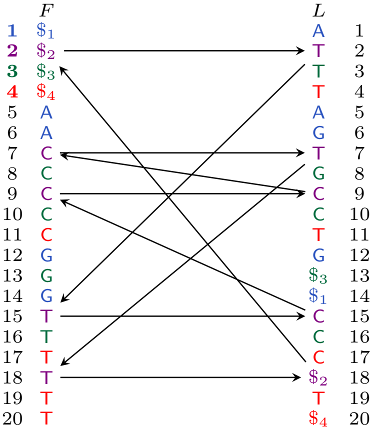
Reversibility and LF mapping

$\mathcal{S}' = \{GGAA\$_1, TCCT\$_2, GCCT\$_3, TTCT\$_4\}$

- LF Mapping: the i-th occurrence of $c \in \Sigma$ in L corresponds to the i-th occurrence of $c \in \Sigma$ in F

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

$S_2 = \quad TCCT\$_2$



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

Cycle decomposition of π_{LF}: (**2** 15 9 7 18)

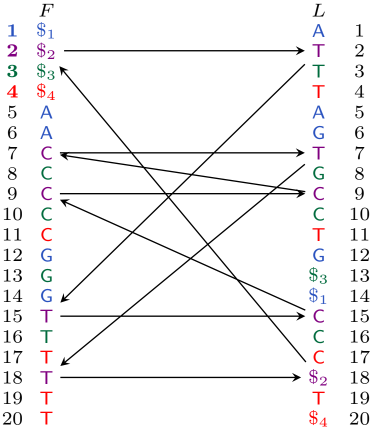
Reversibility and LF mapping

$\mathcal{S}' = \{GGAA\$_1, TCCT\$_2, GCCT\$_3, TTCT\$_4\}$

- LF Mapping:
  the i-th occurrence of $c \in \Sigma$ in L corresponds to the i-th occurrence of $c \in \Sigma$ in F

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

$S_2 = \quad TCCT\$_2$



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

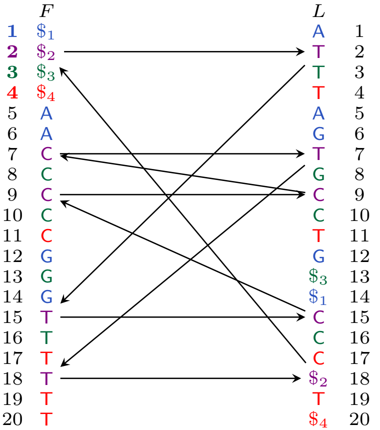
Cycle decomposition of π_{LF}: (**1** 5 6 12 14) (**2** 15 9 7 18) (**3** 16 10 8 13) (**4** 17 11 19 20)

Reversibility and LF mapping

- LF Mapping: the i-th occurrence of $c \in \Sigma$ in L corresponds to the i-th occurrence of $c \in \Sigma$ in F
- For all i, the symbol $F[i]$ (circularly) follows $L[i]$ in the original (corresponding) string.

S_2 can be removed by using (**2** 15 9 7 18)

$\mathcal{S}' = \{GGAA\$_1, TCCT\$_2, GCCT\$_3, TTCT\$_4\}$



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

Cycle decomposition of π_{LF}: (**1** 5 6 12 14) (**2** 15 9 7 18) (**3** 16 10 8 13) (**4** 17 11 19 20)

Reversibility with SAP-heuristics

$\mathcal{S}' = \{GGAA\$_1, TCCT\$_2, GCCT\$_3, TTCT\$_4\}$

- LF Mapping:
  the i-th occurrence of $c \in \Sigma$ in L corresponds to the i-th occurrence of $c \in \Sigma$ in F;

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

	F	plusBWT	
1	$\$_?$	A	1
2	$\$_?$	T	2
3	$\$_?$	T	3
4	$\$_?$	T	4
5	A	A	5
6	A	G	6
7	C	G	7
8	C	T	8
9	C	T	9
10	C	C	10
11	C	C	11
12	G	G	12
13	G	$\$_3$	13
14	G	$\$_1$	14
15	T	C	15
16	T	C	16
17	T	C	17
18	T	$\$_2$	18
19	T	T	19
20	T	$\$_4$	20

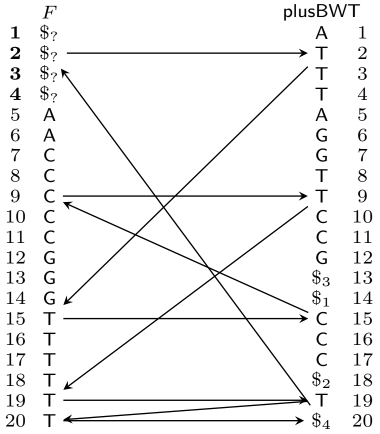
$$\pi_{LF} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 \\ 5 & 15 & 16 & 17 & 6 & 12 & 13 & 18 & 19 & 7 & 8 & 14 & ? & ? & 9 & 10 & 11 & ? & 20 & ? \end{pmatrix}$$

Reversibility with SAP-heuristics

$\mathcal{S}' = \{GGAA\$_1, TCCT\$_2, GCCT\$_3, TTCT\$_4\}$

- **LF Mapping:**
  the i-th occurrence of $c \in \Sigma$ in L corresponds to the i-th occurrence of $c \in \Sigma$ in F;

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



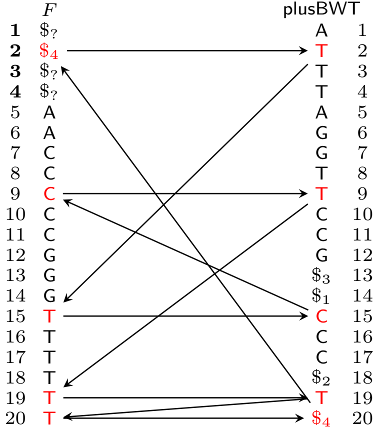
$$\pi_{LF} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 \\ 5 & 15 & 16 & 17 & 6 & 12 & 13 & 18 & 19 & 7 & 8 & 14 & ? & ? & 9 & 10 & 11 & ? & 20 & ? \end{pmatrix}$$

Reversibility with SAP-heuristics

$\mathcal{S}' = \{GGAA\$_1, TCCT\$_2, GCCT\$_3, TTCT\$_4\}$

- **LF Mapping**: the i-th occurrence of $c \in \Sigma$ in L corresponds to the i-th occurrence of $c \in \Sigma$ in F;
- For all i, the symbol $F[i]$ (circularly) follows $L[i]$ in the original (corresponding) string.

$S_4 = TTCT\$_4$



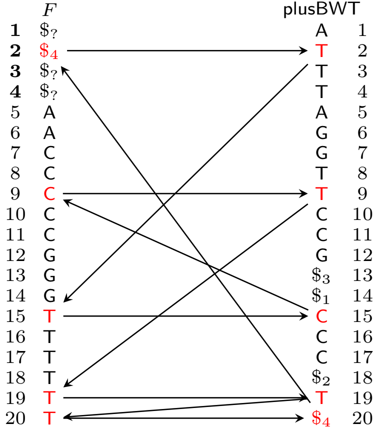
$$\pi_{LF} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 \\ 5 & 15 & 16 & 17 & 6 & 12 & 13 & 18 & 19 & 7 & 8 & 14 & ? & ? & 9 & 10 & 11 & ? & 20 & \mathbf{2} \end{pmatrix}$$

Reversibility with SAP-heuristics

$\mathcal{S}' = \{GGAA\$_1, TCCT\$_2, GCCT\$_3, TTCT\$_4\}$

- LF Mapping: the i-th occurrence of $c \in \Sigma$ in L corresponds to the i-th occurrence of $c \in \Sigma$ in F;
- For all i, the symbol $F[i]$ (circularly) follows $L[i]$ in the original (corresponding) string.

$S_4 = TTCT\$_4$



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

Permutation $\pi_{\mathcal{S}'}$

$$\pi_{LF} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & \overset{\$_3}{13} & \overset{\$_1}{14} & 15 & 16 & 17 & \overset{\$_2}{18} & 19 & \overset{\$_4}{20} \\ 5 & 15 & 16 & 17 & 6 & 12 & 13 & 18 & 19 & 7 & 8 & 14 & \mathbf{3} & \mathbf{1} & 9 & 10 & 11 & \mathbf{4} & 20 & \mathbf{2} \end{pmatrix} \qquad \pi_{\mathcal{S}'} = \begin{pmatrix} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} \\ \mathbf{1} & \mathbf{4} & \mathbf{3} & \mathbf{2} \end{pmatrix}$$

Cycle decomposition of π_{LF}:

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

- The permutation $\pi_{\mathcal{S}'}$ gives the indices of the end-markers in column F
- Having $\pi_{\mathcal{S}'}$, it is possible to decode (and/or remove) groups of strings, without decoding the entire string collection
- We can compute $\pi_{\mathcal{S}'}$ while swapping symbols during the BWT construction

Input order-preserving with SAP-heuristics

$GGAA\$_1$
$TCCT\$_2$
$GCCT\$_3$
$TTCT\$_4$

I iter.		II iter.		III iter.		IV iter.	
A	\$	A	\$	A	\$	A	\$
T	\$	T	\$	T	\$	T	\$
T	\$	T	\$	T	\$	T	\$
T	\$	T	\$	T	\$	T	\$
		A	A\$	A	A\$	A	A\$
		C	T\$	G	AA\$	G	AA\$
		C	T\$	T	CT\$	G	CCT\$
		C	T\$	C	CT\$	T	CCT\$
				C	CT\$	T	CT\$
				C	T\$	C	CT\$
				C	T\$	C	CT\$
				C	T\$	G	GAA\$
						C	T\$
						C	T\$
						C	T\$
						T	TCT\$

...

$\pi_{\mathcal{S}'}$: 1 | 4 | 3 | 2

Conclusion and future work

- We defined a class $\mathfrak{S}_{\mathcal{S}}$ of transformed strings where a different *adaptive* alphabet ordering is applied to symbols in SAP-intervals, while maintaining the reversibility property
- We introduced new heuristics in $\mathfrak{S}_{\mathcal{S}}$ that tend to minimize the number of runs while computing the BWT string
- In the experiments, the new heuristics show a reduction of runs with a negligible overhead, and improve on the previously-introduced ones by Cox et al.
- We also addressed the problem of returning the input permutation when a heuristic is applied
- We plan to consider other BWT-related data structures that can be affected by symbol swapping

Università di Pisa

Thank you!

veronica.guerrini@unipi.it

Github : github.com/giovannarosone/BCR_LCP_GSA/tree/SAP_heuristics

Funded by
the European Union
NextGenerationEU

MUR

INδAM