On the combinatorics of the BWT of string collections

Zsuzsanna Lipták

University of Verona (Italy)

Sequences in London
Goldsmiths, University of London
11 May 2023

The Burrows-Wheeler-Transform

Ex.: $T = \texttt{banana}$. The BWT is a permutation of T: $\texttt{nnbaaa}$

all rotations (conjugates)		all rotations, sorted
banana		abanan
ananab	$\longrightarrow$	anaban
nanaba	lexicographic order	ananab
anaban		banana
nabana		nabana
abanan		nanaba

Take a string T, list all of its rotations, sort them lexicographically, concatenate last characters: $\text{bwt}(\texttt{banana}) = \texttt{nnbaaa}$

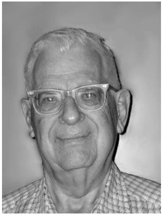
AWARDS & RECOGNITION

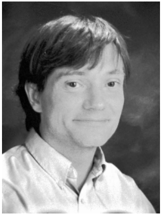
Inventors of BW-transform and the FM-index Receive Kanellakis Award

Michael Burrows, Google; **Paolo Ferragina**, University of Pisa; and **Giovanni Manzini**, University of Pisa, receive the **ACM Paris Kanellakis Theory and Practice Award** for inventing the BW-transform and the FM-index that opened and influenced the field of Compressed Data Structures with fundamental impact on Data Compression and Computational Biology. In 1994, Burrows and his late coauthor David Wheeler published their paper describing revolutionary data compression algorithm based on a reversible transformation of the input—the "Burrows-Wheeler Transform" (BWT). A few years later, Ferragina and Manzini showed that, by orchestrating the BWT with a new set of mathematical techniques and algorithmic tools, it became possible to build a "compressed index," later called the FM-index. The introduction of the BW Transform and the development of the FM-index have had a profound impact on the theory of algorithms and data structures with fundamental advancements.

source: `https://awards.acm.org/kanellakis`

BWT history





- invented by David Wheeler in the 70s as a lossless text compression algorithm
- fully developed and written up together with Michael Burrows in 1994
- appeared as a technical report only, never published
- popularized by Julian Seward's implementation: `bzip` and `bzip2` (1996)

source: Adjeroh, Bell, Mukerjee: The Burrows-Wheeler-Transform, Springer, 2008

Why is the BWT useful in text compression?

rotation	BWT
`he caverns measureless to man, And sank in tumult to a ...`	t
`he caves. It was a miracle of rare device, A sunny pleasure-...`	t
`he dome of pleasure Floated midway on the waves; Where was ...`	t
`he fountain and the caves. It was a miracle of rare device,...`	t
`he green hill athwart a cedarn cover! A savage place! as ...`	t
`he hills, Enfolding sunny spots of greenery. But oh! that ...`	t
`he milk of Paradise.`	t
`he mingled measure From the fountain and the caves. It was a ...`	t
`he on honey-dew hath fed, And drunk the milk of Paradise.  ...`	␣
`he played, Singing of Mount Abora. Could I revive within me ...`	s
`he sacred river ran, Then reached the caverns measureless ...`	t
`he sacred river, ran Through caverns measureless to man ...`	t
`he sacred river. Five miles meandering with a mazy motion ...`	t
`he shadow of the dome of pleasure Floated midway on the waves ...`	T
`he thresher's flail: And mid these dancing rocks at once and ...`	t
`he waves; Where was heard the mingled measure From the ...`	t

Kubla Kahn by Samuel Coleridge

- many `the`'s, some `he,` `she,` `The`

Compression with the BWT

- in original paper: using Move-to-front and Huffman/arithmetic coding

Compression with the BWT

- in original paper: using Move-to-front and Huffman/arithmetic coding
- nowadays: using RLE (runlength-encoding)

Compression with the BWT

- in original paper: using Move-to-front and Huffman/arithmetic coding
- nowadays: using RLE (runlength-encoding)
  - RLE: replace equal-letter-runs by (character, integer)-pair
  - Ex.: $\texttt{bbbbbbbbcaaaaaaaaaaabb} \mapsto (\texttt{b}, 8), (\texttt{c}, 1), (\texttt{a}, 11), (\texttt{b}, 2)$

Compression with the BWT

- in original paper: using Move-to-front and Huffman/arithmetic coding
- nowadays: using RLE (runlength-encoding)
  - RLE: replace equal-letter-runs by (character, integer)-pair
  - Ex.: $\texttt{bbbbbbbbcaaaaaaaaaaabb} \mapsto (\texttt{b}, 8), (\texttt{c}, 1), (\texttt{a}, 11), (\texttt{b}, 2)$
- good if few runs w.r.t. length of string

Compression with the BWT

- in original paper: using Move-to-front and Huffman/arithmetic coding
- nowadays: using RLE (runlength-encoding)
  - RLE: replace equal-letter-runs by (character, integer)-pair
  - Ex.: $\texttt{bbbbbbbbcaaaaaaaaaaabb} \mapsto (\texttt{b}, 8), (\texttt{c}, 1), (\texttt{a}, 11), (\texttt{b}, 2)$
- good if few runs w.r.t. length of string
- Def.: $r(T) = \#$ runs of $\text{bwt}(T)$
  Ex.: $r(\texttt{banana}) = 3$ recall: $\text{bwt}(\texttt{banana}) = \texttt{nnbaaa}$

Compression with the BWT

- in original paper: using Move-to-front and Huffman/arithmetic coding
- nowadays: using RLE (runlength-encoding)
  - RLE: replace equal-letter-runs by (character, integer)-pair
  - Ex.: $\texttt{bbbbbbbbcaaaaaaaaaaabb} \mapsto (\texttt{b}, 8), (\texttt{c}, 1), (\texttt{a}, 11), (\texttt{b}, 2)$
- good if few runs w.r.t. length of string
- Def.: $r(T) = \#$ runs of $\mathrm{bwt}(T)$
  Ex.: $r(\texttt{banana}) = 3$ recall: $\mathrm{bwt}(\texttt{banana}) = \texttt{nnbaaa}$
- for repetitive strings, r is small

The parameter r

Def. String T, $r =$ number of runs of bwt(T).

- size of data structures $\mathcal{O}(r)$
- algorithms' running time ideally a function of r (not of $n = |T|$)
- increasingly used as a repetitiveness measure of T
  - Navarro: "Indexing Highly Repetitive String Collections, Part I: Repetitiveness Measures" [ACM Comp. Surv., 2021]
  - Kempa and Kociumaka: "Resolution of the Burrows-Wheeler Transform Conjecture" [FOCS 2020]
- r (or n/r, the average runlength) is treated as a property of the dataset
- We will argue that for string collections, the parameter r is not well-defined

The BWT of string collections

The BWT of string collections

[Cenzato and L., CPM 2022]

Question: How to compute the BWT of a multiset?
ex. $\mathcal{M} = \{\texttt{ATATG}, \texttt{TGA}, \texttt{ACG}, \texttt{ATCA}, \texttt{GGA}\}$

- Three fundamentally different approaches (with variations)
- These result in different transforms.
- The idea seems to be that it's all the same: not true!

The BWT of string collections

[Cenzato and L., CPM 2022]

Question: How to compute the BWT of a multiset?
ex. $\mathcal{M} = \{\texttt{ATATG}, \texttt{TGA}, \texttt{ACG}, \texttt{ATCA}, \texttt{GGA}\}$

- Three fundamentally different approaches (with variations)
- These result in different transforms.
- The idea seems to be that it's all the same: not true!

The three appraoches are:

1. extended BWT of Mantaci et al.
2. concatenate strings, separating them with different dollars
3. concatenate strings, separating them with same dollar

How to compute the BWT of a multiset of strings?

ex. $\mathcal{M} = \{\texttt{ATATG}, \texttt{TGA}, \texttt{ACG}, \texttt{ATCA}, \texttt{GGA}\}$

variant (our terminology)	result on example	tools
eBWT	CGGGATGTACGTTAAAAA	pfpebwt
dollarEBWT	GGAAACGG$$$TTACTGTAAA	G2BWT, pfpebwt, msbwt
multidolBWT	GAGAAGCG$$$TTATCTGAAA	BCR, ropebwt2, nvSetBWT, Merge-BWT, eGSA, eGAP, bwt-lcp-parallel, gsufsort
concatBWT	$AAGAGGGC$#$TTACTGT$AAA$	BigBWT, tools for single strings
colexBWT	AAAGGCGG$$$TTACTGTAAA	ropebwt2

The different BWT variants

1. eBWT($\mathcal{M}$): the extended BWT of Mantaci et al. (2007)
uses omega-order instead of lexicographical order: e.g. aba $<_\omega$ ab

The different BWT variants

1. eBWT($\mathcal{M}$): the extended BWT of Mantaci et al. (2007)

uses omega-order instead of lexicographical order: e.g. aba $<_\omega$ ab

$T <_\omega S$ if (a) $T^\omega < S^\omega$, or (b) $T^\omega = S^\omega$, $T = U^k, S = U^m$ and $k < m$

The different BWT variants

1. eBWT($\mathcal{M}$): the extended BWT of Mantaci et al. (2007)
uses omega-order instead of lexicographical order: e.g. $\texttt{aba} <_\omega \texttt{ab}$
$T <_\omega S$ if (a) $T^\omega < S^\omega$, or (b) $T^\omega = S^\omega$, $T = U^k, S = U^m$ and $k < m$

- No efficient implementation until 2021
[Boucher, Cenzato, L., Rossi, Sciortino, SPIRE 2021]
- a variation: dollarEBWT($\mathcal{M}$) = eBWT($\{ T_i\$ \ : \ T_i \in \mathcal{M}\}$)
[Diaz-Domingo and Navarro, DCC 2021, CPM 2022]

The different BWT variants

2. multidollarBWT($\mathcal{M}$) = bwt($T_1\$_1 T_2\$_2 \cdots T_k\$_k$), where dollars are smaller than characters from Σ, and $\$_1 < \$_2 < \ldots < \$_k$

The different BWT variants

2. multidollarBWT($\mathcal{M}$) = bwt($T_1\$_1 T_2\$_2 \cdots T_k\$_k$), where dollars are smaller than characters from Σ, and $\$_1 < \$_2 < \ldots < \$_k$

- this is the most commonly used method
- dollars are different only conceptually (break ties by index)
- analogous to Generalized Suffix Tree and Generalized Suffix Array
- equivalent: concatenate without separators, use bitstring marking string beginnings
- a special case:
  colexBWT($\mathcal{M}$) = multidol($\mathcal{M}, \gamma$), where γ is the permutation corresponding to the colexicographic ('reverse lexicographic').

The different BWT variants

3. concatBWT($\mathcal{M}$) = bwt($T_1\$T_2\$\cdots T_k\$\#$), where $\# < \$$

The different BWT variants

3. concatBWT($\mathcal{M}$) = bwt($T_1\$T_2\$\cdots T_k\$\#$), where $\# < \$$

used e.g. in BigBWT. More later.

Interesting intervals

ex. $\mathcal{M} = \{\texttt{ATATG}, \texttt{TGA}, \texttt{ACG}, \texttt{ATCA}, \texttt{GGA}\}$

BWT variant	example
non-sep.based	
eBWT($\mathcal{M}$)	CGGGATGTACGTTAAAAA
separator-based	
dollarEBWT($\mathcal{M}$)	GGAAACGG\$\$\$TTACTGT\$AAA\$
multidolBWT($\mathcal{M}$)	GAGAAGCG\$\$\$TTATCTG\$AAA\$
concatBWT($\mathcal{M}$)	AAGAGGGC\$\$\$TTACTGT\$AAA\$
colexBWT($\mathcal{M}$)	AAAGGCGG\$\$\$TTACTGT\$AAA\$

in color: **interesting intervals**

Interesting intervals

An interval $[i, j]$ is **interesting** if it is the SA-interval of a left-maximal shared suffix U. Then and only then can two separator-based BWTs differ in $[i, j]$.

ex. $\mathcal{M} = \{\mathtt{ATATG}, \mathtt{TGA}, \mathtt{ACG}, \mathtt{ATCA}, \mathtt{GGA}\}$

$\mathtt{A\$\#}$	$\mathtt{G}$
$\mathtt{A\$...}$	$\mathtt{G}$
$\mathtt{A\$...}$	$\mathtt{C}$

U = A$

concBWT

$\mathtt{A\$_2...TG}$	$\mathtt{G}$
$\mathtt{A\$_4...ATC}$	$\mathtt{C}$
$\mathtt{A\$_5...GG}$	$\mathtt{G}$

U = A$

mdolBWT

$\mathtt{A\$_1ATC}$	$\mathtt{C}$
$\mathtt{A\$_2GG}$	$\mathtt{G}$
$\mathtt{A\$_3TG}$	$\mathtt{G}$

U = A$

dolEBWT

Order of shared suffixes

ex. $\mathcal{M} = \{\texttt{ATATG}, \texttt{TGA}, \texttt{ACG}, \texttt{ATCA}, \texttt{GGA}\}$

BWT variant	example	order of shared suffixes
eBWT($\mathcal{M}$)	the extended BWT CGGGATGTACGTTAAAAA	omega-order of strings (mixed in with substrings)
dollarEBWT($\mathcal{M}$)	eBWT($\{T_i\$: T_i \in \mathcal{M}\}$) GGAAACGG\$\$\$TTACTGT\$AAA\$	lexicographic order of strings
multidolBWT($\mathcal{M}$)	bwt($T_1\$_1 T_2\$_2 \cdots T_k\$_k$) GAGAAGCG\$\$\$TTATCTG\$AAA\$	input order of strings
concatBWT($\mathcal{M}$)	bwt($T_1\$ T_2\$ \cdots T_k\$\#$) AAGAGGGC\$\$\$TTACTGT\$AAA\$	lexicographic order of subsequent strings in input
colexBWT($\mathcal{M}$)	multidol($\mathcal{M}, \gamma$), $\gamma =$ colex AAAGGCGG\$\$\$TTACTGT\$AAA\$	colexicographic order

Input order dependence

N.B. multidolBWT and concatBWT depend on the input order!

$\mathcal{M}_1$ = `[ATATG,TGA,ACG,ATCA,GGA]` mdolBWT($\mathcal{M}_1$) = `GAGAAGCG$$$TTATCTG$AAA$`

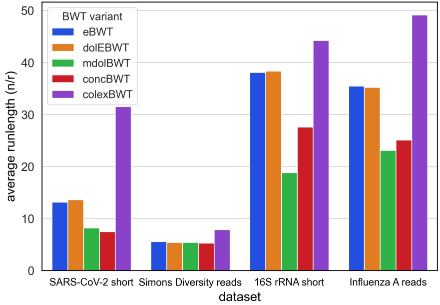
$\mathcal{M}_2$ = `[ACG,ATATG,GGA,TGA,ATCA]` mdolBWT($\mathcal{M}_2$) = `GGAAAGGC$$$TTACTGT$AAA$`

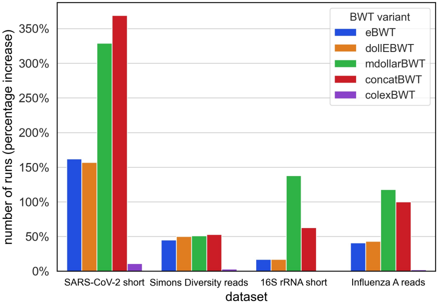
$\mathcal{M}_1$ = `[ATATG,TGA,ACG,ATCA,GGA]` concBWT($\mathcal{M}_1$) = `AAGAGGGC$$$TTACTGT$AAA$`

$\mathcal{M}_2$ = `[ACG,ATATG,GGA,TGA,ATCA]` concBWT($\mathcal{M}_2$) = `AGAGACGG$$$TTACTTG$AAA$`

The parameter r

Results regarding r on four short sequence datasets, of all BWT variants.





Left: average runlength (n/r). Right: number of runs r (percentage increase with respect to the optimal BWT of [Bentley et al., ESA 2020]).

(In each experiment: 500,000 seq.s of length between 50 and 301.)

The different BWT variants

- BWT variants differ significantly among each other ($>11\%$ Hamming distance on some data sets)
- we theoretically explained these differences ("interesting intervals")
- differences especially high on large sets of short sequences
- multidolBWT and concatBWT depend on the input order
- differences extend to parameter r (number of runs of the BWT)

The different BWT variants

- BWT variants differ significantly among each other ($>$ 11% Hamming distance on some data sets)
- we theoretically explained these differences ("interesting intervals")
- differences especially high on large sets of short sequences
- multidolBWT and concatBWT depend on the input order
- differences extend to parameter r (number of runs of the BWT)

We suggest

- to standardize the definition of r (colexBWT or optBWT)
- optBWT now implemented: Cenzato, Guerrini, L., Rosone, DCC 2023 (next)

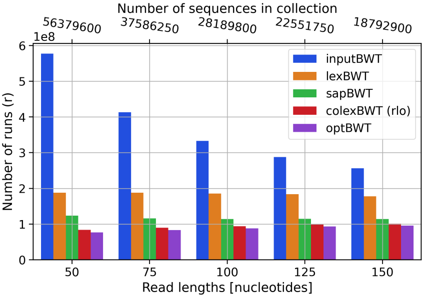
The optimal BWT

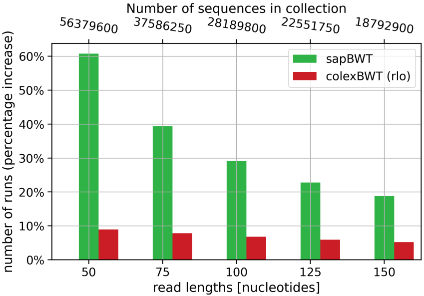
Minimizing the number of runs of the multidollarBWT

[Cenzato, Guerrini, L., Rosone, DCC 2023]

- Bentley et al. [ESA 2020] presented an linear-time algorithm for computing the input order which minimizes r
- We implemented this algorithm, combining it with two BWT construction algorithms (SAIS and BCR)
- negligible computational overhead w.r.t. BWT of input order
- **up to a factor of** 31 reduction of r on real data

optBWT: simulated data





number of runs on simulated datasets of P. Aeruginosa (cov. 450x), for varying read lengths. Left: number of runs. Right: percentage increase of the two heuristics sapBWT and colexBWT with respect to the optimal BWT.

optBWT: real data

data set	number of runs increase compared to optimal BWT (factor and perc.)				resource usage (optBWT)	
	inputBWT	colexBWT (rlo)	sapBWT	lexBWT	RAM (GB)	Time (hh:mm:ss)
1	**4.22** (322.26%)	1.03 (3.48%)	1.53 (53.06%)	1.30 (30.13%)	6.45 (1.02×)	7:18 (1.12×)
2	**14.07** (1306.95%)	1.15 (14.54%)	1.21 (20.75%)	3.52 (252.39%)	8.08 (1.03×)	6:32 (1.15×)
3	**3.65** (264.90%)	1.07 (6.52%)	1.30 (29.63%)	2.07 (107.01%)	11.15 (1.04×)	18:29 (1.26×)
4	**5.17** (416.52%)	1.04 (4.38%)	1.55 (55.33%)	1.55 (54.87%)	21.03 (1.02×)	22:08 (1.08×)
5	**2.44** (144.36%)	1.05 (5.05%)	1.16 (15.73%)	2.03 (103.35%)	4.31 (1.04×)	2:25:46 (1.28×)
6	**31.49** (3048.66%)	1.04 (4.30%)	1.79 (79.40%)	1.89 (89.17%)	8.86 (1.05×)	1:59:46 (1.39×)
7	**2.13** (112.56%)	1.04 (4.17%)	1.12 (11.89%)	1.96 (96.04%)	34.42 (1.03×)	26:24:18 (1.48×)

Increase in the number of runs compared to the optBWT (left), and resource usage (right). For each BWT, increase factor and the percentage increase (in brackets). Total time and memory for building the optBWT from scratch, and overhead with respect to constructing the inputBWT only (in brackets).

dataset 2: SARS-CoV-2 reads (33 mio. sequences of length 50);

dataset 6: Sindibis virus reads (431 mio. sequences of length 36).

What is concatBWT?

Order matters!

~~$\mathcal{M} = \{\texttt{ATATG}, \texttt{TGA}, \texttt{ACG}, \texttt{ATCA}, \texttt{GGA}\}$~~ $\mathcal{M} = [\texttt{ATATG}, \texttt{TGA}, \texttt{ACG}, \texttt{ATCA}, \texttt{GGA}]$

BWT variant	example	order of shared suffixes
eBWT($\mathcal{M}$)	the extended BWT CGGGATGTACGTTAAAAA	omega-order of strings (mixed in with substrings)
dollarEBWT($\mathcal{M}$)	eBWT($\{T_i\$: T_i \in \mathcal{M}\}$) GGAAACGG\$\$\$TTACTGT\$AAA\$	lexicographic order of strings
multidolBWT($\mathcal{M}$)	bwt($T_1\$_1 T_2\$_2 \cdots T_k\$_k$) GAGAAGCG\$\$\$TTATCTG\$AAA\$	input order of strings
concatBWT($\mathcal{M}$)	bwt($T_1\$ T_2\$ \cdots T_k\$\#$) AAGAGGGC\$\$\$TTACTGT\$AAA\$	lexicographic order of subsequent strings in input
colexBWT($\mathcal{M}$)	multidol($\mathcal{M}, \gamma$), γ = colex AAAGGCGG\$\$\$TTACTGT\$AAA\$	colexicographic order

In the *k*-prefix (shared suffix: \$) of the BWT we see the output order.

What is the output order of the concatBWT?

[Cenzato, L., Masillo, Rossi, forthcoming]

$\mathcal{M} = [\texttt{ATATG}, \texttt{TGA}, \texttt{ACG}, \texttt{ATCA}, \texttt{GGA}] \quad \mapsto \quad \texttt{ATATG\$TGA\$ACG\$ATCA\$GGA\$\#}$

concatBWT$(\mathcal{M}) = \text{BWT}(\texttt{ATATG\$TGA\$ACG\$ATCA\$GGA\$\#})$

Map strings to their lexicographic rank:

$$\begin{array}{lcl} \texttt{ACG} & \mapsto & \texttt{a} \\ \texttt{ATATG} & \mapsto & \texttt{b} \\ \texttt{ATCA} & \mapsto & \texttt{c} \\ \texttt{GGA} & \mapsto & \texttt{d} \\ \texttt{TGA} & \mapsto & \texttt{e} \end{array}$$

$$\mathcal{M} = \underbrace{\texttt{ATATG}}_{\texttt{b}}\$\underbrace{\texttt{TGA}}_{\texttt{e}}\$\underbrace{\texttt{ACG}}_{\texttt{a}}\$\underbrace{\texttt{ATCA}}_{\texttt{c}}\$\underbrace{\texttt{GGA}}_{\texttt{d}}\$\# \quad \mapsto \quad \texttt{beacd\#}.$$

What is the output order of the concatBWT?

$$\mathcal{M} = \underbrace{\texttt{ATATG}}_{\texttt{b}}\$\underbrace{\texttt{TGA}}_{\texttt{e}}\$\underbrace{\texttt{ACG}}_{\texttt{a}}\$\underbrace{\texttt{ATCA}}_{\texttt{c}}\$\underbrace{\texttt{GGA}}_{\texttt{d}}\$\# \quad \mapsto \quad \texttt{beacd\#}.$$

index	concatBWT	rotation
23	A	$#ATATG$TGAACGATCA$GGA
10	A	ACGATCAGGA#ATATG$TGA
14	G	$ATCA$GGA$#ATATG$TGA$ACG
19	A	GGA#ATATGTGAACG$ATCA
6	G	TGAACG$ATCA$GGA$#ATATG
...	...	...

input: b e a c d # **output:** d e a c b

What is the output order of the concatBWT?

input: `b e a c d #` **output:** `d e a c b`

What is the output order of the concatBWT?

input: `b e a c d #` **output:** `d e a c b`

This is the BWT of the metacharacter-string! (almost)

BWT(`beacd#`) = `de#acb` $\rightsquigarrow$ `deacb`

output = BWT(**input#**) (remove the # from the output)

What is the output order of the concatBWT?

- the (output order of the) concatBWT is the BWT of the meta-string of the input
- for many datasets, the concatBWT and the multidollarBWT will differ
- the concatBWT cannot produce all BWT variants
- only those for which there exists a position into which the # can be inserted s.t. it becomes the BWT of some meta-string
- which are these? next

When a dollar makes a BWT

When a dollar makes a BWT

[Giuliani, L., Masillo, Rizzi, TCS, 2021]

Question: Given a word W, can we insert \$ somewhere to make it a BWT?

When a dollar makes a BWT

[Giuliani, L., Masillo, Rizzi, TCS, 2021]

Question: Given a word W, can we insert \$ somewhere to make it a BWT?

Ex.: $W = \texttt{annbaa}$.

0	`$annbaa`	-
1	`a$nnbaa`	-
2	`an$nbaa`	-
3	`ann$baa`	-
4	`annb$aa`	bwt(`banana$`)
5	`annba$a`	-
6	`annbaa$`	bwt(`nabana$`)

`annbaa`: yes ✓

When a dollar makes a BWT

[Giuliani, L., Masillo, Rizzi, TCS, 2021]

Question: Given a word W, can we insert \$ somewhere to make it a BWT?

Ex.: $W =$ `annbaa`.

0	`$annbaa`	-
1	`a$nnbaa`	-
2	`an$nbaa`	-
3	`ann$baa`	-
4	`annb$aa`	bwt(`banana$`)
5	`annba$a`	-
6	`annbaa$`	bwt(`nabana$`)

`annbaa`: yes ✓

Ex.: $W =$ `banana`.

0	`$banana`	-
1	`b$anana`	-
2	`ba$nana`	-
3	`ban$ana`	-
4	`bana$na`	-
5	`banan$a`	-
6	`banana$`	-

`banana`: no ✗

Our algorithm

- Simple algorithm: for every i, $0 \leq i < n$, try reversing the BWT: $\mathcal{O}(n^2)$ time
- Our algorithm: $\mathcal{O}(n \log n)$ time
- def.: π_i standard permutation of W with \$ in position i
- idea: compute π_{i+1} directly from π_i in $\mathcal{O}(\log n)$ time
- smart use of splay trees for maintaining permutations

Our algorithm

Lemma: We can get π_{i+1} from π_i with one transposition:

$$\pi_{i+1} = (\pi_i(i), \pi_i(i+1)) \circ \pi_i \underset{\text{\$ is in position } i}{=} (0, \pi_i(i+1)) \circ \pi_i$$

Lemma

1. Transposition of elements in **distinct** cycles **merges** the two cycles
2. Transposition of elements in the **same** cycle **splits** the cycle

Our algorithm

1. Transposition of elements in **distinct** cycles **merges** the two cycles

   $\left(\begin{smallmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ 0 & 5 & 6 & 4 & 1 & 2 & 3 \end{smallmatrix}\right) = (0)(1, 5, 2, 6, 3, 4)$

   $\left(\begin{smallmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ 5 & 0 & 6 & 4 & 1 & 2 & 3 \end{smallmatrix}\right) = (0, 5, 2, 6, 3, 4, 1)$

2. Transposition of elements in the **same** cycle **splits** the cycle

   $\left(\begin{smallmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ 5 & 0 & 6 & 4 & 1 & 2 & 3 \end{smallmatrix}\right) = (0, 5, 2, 6, 3, 4, 1)$

   $\left(\begin{smallmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ 5 & 6 & 0 & 4 & 1 & 2 & 3 \end{smallmatrix}\right) = (0, 5, 2)(6, 3, 4, 1)$

Our algorithm

Ex.: Algorithm **findNicePositions(W)** on $W = \texttt{annbaa}$:

0	`$annbaa`	$\pi_0 = \begin{pmatrix} 0&1&2&3&4&5&6 \\ 0&1&5&6&4&2&3 \end{pmatrix} = (0)(1)(2,5)(3,6)(4)$	merge
1	`a$nnbaa`	$\pi_1 = \begin{pmatrix} 0&1&2&3&4&5&6 \\ 1&2&5&6&4&2&3 \end{pmatrix} = (0,1)(2,5)(3,6)(4)$	merge
2	`an$nbaa`	$\pi_2 = \begin{pmatrix} 0&1&2&3&4&5&6 \\ 1&5&0&6&4&2&3 \end{pmatrix} = (0,1,5,2)(3,6)(4)$	merge
3	`ann$baa`	$\pi_3 = \begin{pmatrix} 0&1&2&3&4&5&6 \\ 1&5&6&0&4&2&3 \end{pmatrix} = (0,1,5,2,6,3)(4)$	merge
4	`annb$aa`	$\boxed{\pi_4 = \begin{pmatrix} 0&1&2&3&4&5&6 \\ 1&5&6&4&0&2&3 \end{pmatrix} = (0,1,5,2,6,3,4)}$	split
5	`annba$a`	$\pi_5 = \begin{pmatrix} 0&1&2&3&4&5&6 \\ 1&5&6&4&2&0&3 \end{pmatrix} = (0,1,5)(2,6,3,4)$	merge
6	`annbaa$`	$\boxed{\pi_6 = \begin{pmatrix} 0&1&2&3&4&5&6 \\ 1&5&6&4&2&3&0 \end{pmatrix} = (0,1,5,3,4,2,6)}$	

Conclusions

- there are different ways of computing the BWT of a string collection
- these are non-equivalent
- several are input-order dependent (in part. multidollarBWT and concatBWT)
- the number of runs r varies significantly
- for the multidollarBWT, optBWT minimzes r, and has been implemented
- the concatBWT is more restrictive ("bwt of input order")
- definition of r should be standardized (optBWT or colexBWT)

Papers

- D. Cenzato and Zs. Lipták: A theoretical and experimental analysis of BWT variants for string collections, CPM 2022.
`github.com/davidecenzato/BWT-variants-for-string-collections`
- D. Cenzato, V. Guerrini, Zs. Lipták, and G. Rosone: Computing the optimal BWT for very large string collections, DCC 2023.
`github.com/davidecenzato/optimalBWT`
- S. Giuliani, Zs. Lipták, F. Masillo, R. Rizzi: When a dollar makes a BWT, Theor. Comput. Sc., 2021.

Acknowledgements (co-authors of this work)

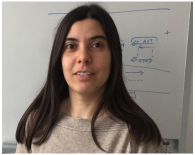
Davide Cenzato
(Univ. of Venice)

Sara Giuliani
(Univ. of Verona)

Veronica Guerrini
(Univ. of Pisa)

Francesco Masillo
(Univ. of Verona)

Romeo Rizzi
(Univ. of Verona)



Giovanna Rosone
(Univ. of Pisa)

Massimiliano Rossi
(Univ. of Florida)

Thank you for your attention!

email: **zsuzsanna.liptak@univr.it**