Dollar or no dollar, that is the question

New combinatorial results on the Burrows-Wheeler-Transform

Zsuzsanna Lipták

University of Verona (Italy)

Seminario de Teoria y Datos
PUC Chile, Santiago, Nov. 4, 2022

Part I:

Introduction

The BWT

The BWT

source: group-media.mercedes-benz.com

The BWT

source: group-media.mercedes-benz.com

(BWT here stands for: Best Water Technology)

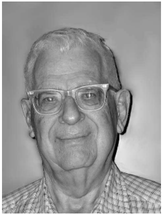
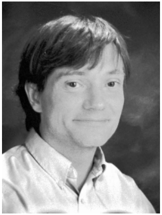
The Burrows-Wheeler-Transform

Ex.: T = banana. The BWT is a permutation of T: nnbaaa

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

Take a string (word) T, list all of its rotations, sort them lexicographically, concatenate last characters: bwt(T).

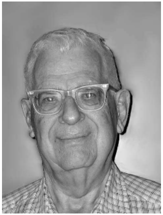
BWT history

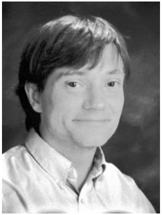



- invented by David Wheeler in the 70s
  as a lossless text compression algorithm
- fully developed and written up together with Michael Burrows in 1994
- appeared as a technical report only, never published
- popularized by Julian Seward's implementation: `bzip` and `bzip2` (1996)

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

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 can the BWT be useful in text compression?

BWT-matrix (F = first column, L = last column)

	F					L
0	a	b	a	n	a	n
1	a	n	a	b	a	n
2	a	n	a	n	a	b
3	b	a	n	a	n	a
4	n	a	b	a	n	a
5	n	a	n	a	b	a

Why can the BWT be useful in text compression?

BWT-matrix (F = first column, L = last column)

	F		L
0	a	bana	n
1	a	naba	n
2	a	nana	b
3	b	anan	a
4	n	aban	a
5	n	anab	a

- **Obs. 1:** F = all characters of T in lex. order: `aaabnn`

Why can the BWT be useful in text compression?

BWT-matrix (F = first column, L = last column)

	F	L
0	abana	n
1	anaba	n
2	anana	b
3	banan	a
4	naban	a
5	nanab	a

- **Obs. 1:** F = all characters of T in lex. order: aaabnn
- **Obs. 2:** for all i: L_i precedes F_i in T:

  $T = \underset{0\,1\,2\,3\,4\,5}{\texttt{banana}}$

Why can the BWT be useful in text compression?

BWT-matrix (F = first column, L = last column)

	F	L
0	abana	n
1	anaba	n
2	anana	b
3	banan	a
4	naban	a
5	nanab	a

- **Obs. 1:** F = all characters of T in lex. order: `aaabnn`
- **Obs. 2:** for all i: L_i precedes F_i in T:
  $T = \underset{0\,1\,2\,3\,4\,5}{\texttt{banana}}$
- **Obs. 3:** all occurrences of a substring appear in consecutive rows

Why can the BWT be useful in text compression?

- **Obs. 1:** F = all characters of T in lex. order.
- **Obs. 2:** L_i precedes F_i in T
- **Obs. 3:** all occurrences of a substring appear in consecutive rows

Why can the BWT be useful in text compression?

- **Obs. 1:** F = all characters of T in lex. order.
- **Obs. 2:** L_i precedes F_i in T
- **Obs. 3:** all occurrences of a substring appear in consecutive rows

Ex.: T = `banana` has 2 occurrences of the pattern `ana`

2 occ's of `ana`

```
abanan
anaban
ananab
banana
nabana
nanaba
```

Why can the BWT be useful in text compression?

- **Obs. 1:** F = all characters of T in lex. order.
- **Obs. 2:** L_i precedes F_i in T
- **Obs. 3:** all occurrences of a substring appear in consecutive rows

Ex.: T = `banana` has 2 occurrences of the pattern `ana`

2 occ's of `ana`	2 occ's of `na` preceded by `a`
`abanan`	`abanan`
`anaban`	`anaban`
`ananab`	`ananab`
`banana`	`banana`
`nabana`	`nabana`
`nanaba`	`nanaba`

Why can the BWT be useful in text compression?

- **Obs. 1:** F = all characters of T in lex. order.
- **Obs. 2:** L_i precedes F_i in T
- **Obs. 3:** all occurrences of a substring appear in consecutive rows

Ex.: T = `banana` has 2 occurrences of the pattern `ana`

2 occ's of `ana`	2 occ's of `na` preceded by `a`	2 occ's of `a` preceded by `n`
abanan	abanan	abanan
anaban	anaban	anaban
ananab	ananab	ananab
banana	banana	banana
nabana	nabana	nabana
nanaba	nanaba	nanaba

Why can the BWT be useful in text compression?

- **Obs. 1:** F = all characters of T in lex. order.
- **Obs. 2:** L_i precedes F_i in T
- **Obs. 3:** all occurrences of a substring appear in consecutive rows

Ex.: T = `banana` has 2 occurrences of the pattern `ana`

2 occ's of `ana`	2 occ's of `na` preceded by `a`	2 occ's of `a` preceded by `n`
abanan	abanan	abanan
anaban	anaban	anaban
ananab	ananab	ananab
banana	banana	banana
nabana	nabana	nabana
nanaba	nanaba	nanaba

So: we get a run of `a`'s of length 2, and a run of `n`'s of length 2

Why can the BWT be useful in text compression?

- **Obs. 1:** F = all characters of T in lex. order.
- **Obs. 2:** L_i precedes F_i in T
- **Obs. 3:** all occurrences of a substring appear in consecutive rows

Ex.: T = banana has 2 occurrences of the pattern ana

2 occ's of ana	2 occ's of na preceded by a	2 occ's of a preceded by n
abanan	abanan	abanan
anaban	anaban	anaban
ananab	ananab	ananab
banana	banana	banana
nabana	nabana	nabana
nanaba	nanaba	nanaba

So: we get a run of a's of length 2, and a run of n's of length 2 (2 = no. occ's).

Of course, things are a bit more complicated:

Of course, things are a bit more complicated:

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.: bbbbbbbbcaaaaaaaaaaabb $\mapsto$ $(\mathtt{b}, 8), (\mathtt{c}, 1), (\mathtt{a}, 11), (\mathtt{b}, 2)$

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.: bbbbbbbbcaaaaaaaaaaabb $\mapsto (\mathtt{b}, 8), (\mathtt{c}, 1), (\mathtt{a}, 11), (\mathtt{b}, 2)$
- 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 bwt(T)

  Ex.: $r(\texttt{banana}) = 3$ recall: 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 bwt(T)
  Ex.: $r(\texttt{banana}) = 3$ recall: bwt$(\texttt{banana}) = \texttt{nnbaaa}$
- for repetitive strings, r is small

BWT magic

The BWT ...

- requires same space as T in bits: $n \log \sigma$ bits $\sigma =$ alphabetsize
  (suffix array: $n \log n$ bits, suffix tree: much more—still $\mathcal{O}(n)$) $n = |T|$
- easier to compress than T, if T repetitive
- very fast (!!!) pattern matching (most basic problem on strings)
- computable in linear time $\mathcal{O}(n)$
- reversible in linear time $\mathcal{O}(n)$ nnbaaa,3 $\mapsto$ banana
- can replace text (suffix array, suffix tree: no)

Compressed data structures for strings

Data structures based on the BWT:

- FM-index [Ferragina and Manzini, FOCS 2000]
- RLFM-index [Mäkinen and Navarro, CPM 2005]
- r-index [Gagie et al, JACM 2020; Bannai et al. TCS 2020]
- some recent developments on r-index [Rossi et al. JCB 2022; Giuliani et al. SEA 2022; Cobas et al. CPM 2021; Boucher et al. SPIRE 2021]

Some tools in bioinformatics (aligners):

- bwa [Durbin and Li, 2009] ca. 41,000 cit.
- bowtie [Langmead and Salzberg, 2010] ca. 36,000 cit.
- soap2 [Li et al., 2009]
- ...

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
- some papers on r:
  - Manzini: "An analysis of the Burrows-Wheeler-Transform" [JACM 2001]
  - Kempa and Kociumaka: "Resolution of the Burrows-Wheeler Transform Conjecture" [FOCS 2020]
  - Navarro: "Indexing Highly Repetitive String Collections, Part I: Repetitiveness Measures" [ACM Comp. Surv., 2021]
  - Mantaci et al.: "Measuring the clustering effect of BWT via RLE" [TCS 2017]

BWT from a combinatorial perspective

- special case of the Gessel-Reutenauer-bijection [Crochemore, Désarménien, Perrin, 2004]
- introduction of the extended BWT (eBWT), a generalization of the BWT to multisets of strings [Mantaci et al. 2007]
- strings T with fully clustering BWTs (e.g. $\text{bwt}(T) = \texttt{bbbbaaccc}$)
  - full characterization for $\sigma = 2$ [Mantaci et al., 2003]
  - partial characterization for $\sigma > 2$ [Puglisi et al., 2008]
  - characterization via interval exchanges [Ferenczi et al., 2013]
- fixpoints of the BWT [Mantaci et al., 2017]
- characterization of BWT images [Likhomanov and Shur, 2011]

Good overview: Rosone and Sciortino: "The Burrows-Wheeler Transform between Data Compression and Combinatorics on Words." [CiE 2013]

- two research communities working on the BWT
- (1) data structures and algorithms on strings and
  (2) combinatorics on words
- little interaction until ...

Dagstuhl workshop "25 years of the Burrows-Wheeler-Transform" (2019)
organized by T. Gagie, G. Manzini, G. Navarro, J. Stoye

But: The two communities use slightly different definitions of the BWT:

- Data Structures and Algorithms on Strings:
  It is assumed that each string terminates with an end-of-string character (denoted \$, smaller than all others)

  $T = \texttt{banana\$}$
- Combinatorics on Words: no such assumption $T = \texttt{banana}$

But: The two communities use slightly different definitions of the BWT:

- Data Structures and Algorithms on Strings:
  It is assumed that each string terminates with an end-of-string character (denoted \$, smaller than all others)

  $T = \texttt{banana\$}$
- Combinatorics on Words: no such assumption $T = \texttt{banana}$

Part II:

Dollar or no dollar, that is the question

1. The transform itself

Different transforms

banana	banana$
abanan	$banana
anaban	a$banan
ananab	ana$ban
banana	anana$b
nabana	banana$
nanaba	na$bana
	nana$ba
nnbaaa	annb$aa

Different transforms

	without dollar (banana)	with dollar (banana$)
the transform	nnbaaa	annb$aa

Different transforms

	without dollar (banana)	with dollar (banana$)
the transform	nnbaaa	annb$aa
remove $	nnbaaa	annbaa

Different transforms

	without dollar (banana)	with dollar (banana$)
the transform	nnbaaa	annb$aa
remove $	nnbaaa	annbaa
# runs r	3	4

Different transforms

	without dollar (banana)	with dollar (banana$)
the transform	nnbaaa	annb$aa
remove $	nnbaaa	annbaa
# runs r	3	4

- **Thm.** There exist strings for which the difference in r is $\Theta(\log n)$.
  [Giuliani, Inenaga, L., Sciortino, 2022, forthcoming]

Different transforms

	without dollar (banana)	with dollar (banana$)
the transform	nnbaaa	annb$aa
remove $	nnbaaa	annbaa
# runs r	3	4

- **Thm.** There exist strings for which the difference in r is $\Theta(\log n)$.
  [Giuliani, Inenaga, L., Sciortino, 2022, forthcoming]
- This is asymptotically tight: here $r = O(1)$, and upper bound is $\mathcal{O}(\log r \log n)$. [Akagi, Funakoshi, Inenaga, 2021]

Different transforms

[Giuliani, Inenaga, L., Sciortino, 2022, forthcoming]

Thm. There exist strings for which the difference in r is $\Theta(\log n)$.

- $r(T\$)$ increases by $\log n$: Fibonacci words of even order $T = Fib(2k), r(T) = 2, r(T\$) = 2k - 1$

  ex.:
  $r(Fib(8)) = 2, r(Fib(8)\$) = 7$
  $r(Fib(12)) = 2, r(Fib(12)\$) = 11$

- $r(T\$)$ decreases by $\log n$: Fibonacci words of odd order without the first character $T = Fib(2k+1)[1:], r(T) = 2k, r(T\$) = 5$

  ex:
  $r(Fib(13)[1:]) = 12, r(Fib(13)[1:]\$) = 5$
  $r(Fib(15)[1:]) = 14, r(Fib(15)[1:]\$) = 5$

Different transforms

[Giuliani, Inenaga, L., Sciortino, 2022, forthcoming]

Thm. There exist strings for which the difference in r is $\Theta(\log n)$.

- $r(T\$)$ increases by $\log n$: Fibonacci words of even order $T = Fib(2k), r(T) = 2, r(T\$) = 2k - 1$

  ex.:
  $r(Fib(8)) = 2, r(Fib(8)\$) = 7$
  $r(Fib(12)) = 2, r(Fib(12)\$) = 11$

- $r(T\$)$ decreases by $\log n$: Fibonacci words of odd order without the first character $T = Fib(2k+1)[1:], r(T) = 2k, r(T\$) = 5$

  ex:
  $r(Fib(13)[1:]) = 12, r(Fib(13)[1:]\$) = 5$
  $r(Fib(15)[1:]) = 14, r(Fib(15)[1:]\$) = 5$

- both additive and multiplicative difference

2. BWT construction

BWT construction

Most BWT construction algorithms first construct the Suffix Array (SA), then construct the BWT from the SA, using: $L_i = T_{SA[i]-1}$ (recall Obs. 2).

ex. $T = \underset{0\,1\,2\,3\,4\,5\,6}{\texttt{banana\$}}$.

SA	
6	$
5	a$
3	ana$
1	anana$
0	banana$
4	na$
2	nana$

BWT construction

Most BWT construction algorithms first construct the Suffix Array (SA), then construct the BWT from the SA, using: $L_i = T_{SA[i]-1}$ (recall Obs. 2).

ex. $T = \underset{0123456}{\texttt{banana\$}}$.

SA	
6	$
5	a$
3	ana$
1	anana$
0	banana$
4	na$
2	nana$

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

BWT construction

Most BWT construction algorithms first construct the Suffix Array (SA), then construct the BWT from the SA, using: $L_i = T_{SA[i]-1}$ (recall Obs. 2).

ex. $T = \underset{0123456}{\texttt{banana\$}}$.

SA	
6	$
5	a$
3	ana$
1	anana$
0	banana$
4	na$
2	nana$

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

Thus: SA-construction in $\mathcal{O}(n)$ time $\Rightarrow$ BWT-construction in $\mathcal{O}(n)$ time.

BWT construction without dollar

- This works fine if there is a \$.
- What if there is no dollar?

BWT construction without dollar

Problem 1:

banana
0 1 2 3 4 5

SA	
5	a
3	ana
1	anana
0	banana
4	na
2	nana

nnbaaa ✓

BWT construction without dollar

Problem 1:

banana
0 1 2 3 4 5

SA		SA	L
5	a	5	abanan
3	ana	3	anaban
1	anana	1	ananab
0	banana	0	banana
4	na	4	nabana
2	nana	2	nanaba

nnbaaa ✓

BWT construction without dollar

Problem 1:

banana
0 1 2 3 4 5

anaban
0 1 2 3 4 5

SA		SA		L
5	a	5	abanan	n
3	ana	3	anaban	n
1	anana	1	ananab	b
0	banana	0	banana	a
4	na	4	nabana	a
2	nana	2	nanaba	a

nnbaaa ✓

BWT construction without dollar

Problem 1:

banana
0 1 2 3 4 5

SA	
5	a
3	ana
1	anana
0	banana
4	na
2	nana

nnbaaa ✓

SA	L
5	abanan
3	anaban
1	ananab
0	banana
4	nabana
2	nanaba

anaban
0 1 2 3 4 5

SA	
2	aban
4	an
0	anaban
3	ban
5	n
1	naban

nbnaaa ✗

BWT construction without dollar

Problem 1:

banana
0 1 2 3 4 5

SA		SA	L
5	a	5	abanan
3	ana	3	anaban
1	anana	1	ananab
0	banana	0	banana
4	na	4	nabana
2	nana	2	nanaba

nnbaaa ✓

anaban
0 1 2 3 4 5

SA		SA	L
2	aban	2	abanan
4	an	4	ananab
0	anaban	0	anaban
3	ban	3	banana
5	n	5	nabana
1	naban	1	nabana

nbnaaa ✗

BWT construction without dollar

Problem 1:

banana
0 1 2 3 4 5

SA		SA	L
5	a	5	abanan
3	ana	3	anaban
1	anana	1	ananab
0	banana	0	banana
4	na	4	nabana
2	nana	2	nanaba

nnbaaa ✓

anaban
0 1 2 3 4 5

SA		SA	L
2	aban	2	abanan
4	an	4	ananab
0	anaban	0	anaban
3	ban	3	banana
5	n	5	nabana
1	naban	1	nabana

nbnaaa ✗

N.B. $suf_i < suf_j \Leftrightarrow conj_i < conj_j$ does not hold in general!

Thus: We need the CA (conjugate array), not the SA!

BWT construction without dollar

Problem 2: If T not primitive, then CA not defined (several identical rotations):

$\underset{0\,1\,2\,3\,4\,5}{\texttt{nanana}} = (\texttt{na})^3$

CA	
1?	anana**n**
3?	anana**n**
5?	anana**n**
0?	nanan**a**
2?	nanan**a**
4?	nanan**a**

Linear-time BWT construction without dollar

- For \$-terminated strings, neither problem exists.
- For Lyndon words (primitive and $<$ all their rotations), neither problem exists.
- All previous BWT-construction algorithms either use \$ or Lyndon rotations.

Our algorithm [Boucher, Cenzato, L., Rossi, Sciortino, SPIRE, 2021]:

- first linear-time BWT-construction algorithm which uses neither \$ nor Lyndon rotations
- adaptation of the SAIS-algorithm for SA-construction [Nong et al., 2011]
- previously, SAIS had been adapted for $T\$$ [Okanohara and Sadakane 2009], and to the bijective BWT [Bannai et al., 2021]

Our algorithm for BWT construction

[Boucher, Cenzato, L., Rossi, Sciortino, SPIRE, 2021]

1. assign circular types to positions
2. sort LMS-substrings
3. assign new names to LMS-substrings
4. construct new string, solve recursively
5. induce CA from relative order of LMS-positions

Step 1

0	1	2	3	4	5
b	a	n	a	n	a
L	S	L	S	L	S
	*		*		*

Step 2

	a			*b*	*n*	
S*	1	3	5			
L				0	2	4
S	5	1	3			
	5	1	3	0	2	4

Step 3

5	a	b	a	A
1	a	n	a	B
3	a	n	a	B

Step 4

0	1	2
A	B	B
S	L	L
*		

A	*B*	
0		
	2	1
0	2	1

Step 5

	a			*b*	*n*	
	5	3	1			
				0	4	2
CA	5	3	1	0	4	2
BWT	n	n	b	a	a	a

BWT without dollar

Implementations of SAIS for conjugate array (cais) for

- BWT without $
- eBWT (extended BWT) (see later)
- BBWT (bijective BWT)
- option for including dollar(s)

See `https://github.com/davidecenzato/cais`

3. BWT of string collections

How to compute the BWT of a multiset of strings?

[Cenzato and L., CPM 2022]

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

It turns out that there are several non-equivalent methods in use:

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 $k < m$
2. dollarEBWT($\mathcal{M}$) $=$ eBWT($\{ T_i\$ \ : \ T_i \in \mathcal{M}\}$)

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. 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 $k < m$
2. dollarEBWT($\mathcal{M}$) $=$ eBWT($\{T_i\$: T_i \in \mathcal{M}\}$)
3. multidolBWT($\mathcal{M}$) $=$ 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

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 $k < m$
2. dollarEBWT($\mathcal{M}$) $=$ eBWT($\{T_i\$ \ : \ T_i \in \mathcal{M}\}$)
3. multidolBWT($\mathcal{M}$) $=$ bwt($T_1\$_1 T_2\$_2 \cdots T_k\$_k$), where dollars are smaller than characters from Σ, and $\$_1 < \$_2 < \ldots < \$_k$
4. concatBWT($\mathcal{M}$) $=$ bwt($T_1\$ T_2\$ \cdots T_k\$\#$), where $\# < \$$

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 $k < m$
2. dollarEBWT($\mathcal{M}$) $=$ eBWT($\{T_i\$ \ : \ T_i \in \mathcal{M}\}$)
3. multidolBWT($\mathcal{M}$) $=$ bwt($T_1\$_1 T_2\$_2 \cdots T_k\$_k$), where dollars are smaller than characters from Σ, and $\$_1 < \$_2 < \ldots < \$_k$
4. concatBWT($\mathcal{M}$) $=$ bwt($T_1\$T_2\$ \cdots T_k\$\#$), where $\# < \$$
5. colexBWT($\mathcal{M}$) $=$ multidol($\mathcal{M}, \gamma$), where γ is the permutation corresponding to the colexicographic ('reverse lexicographic').

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$$$TTACTGTAAA
multidolBWT($\mathcal{M}$)	GAGAAGCG$$$TTATCTGAAA
concatBWT($\mathcal{M}$)	AAGAGGGC$$$TTACTGTAAA
colexBWT($\mathcal{M}$)	AAAGGCGG$$$TTACTGTAAA

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$$$TTACTGTAAA
multidolBWT($\mathcal{M}$)	GAGAAGCG$$$TTATCTGAAA
concatBWT($\mathcal{M}$)	AAGAGGGC$$$TTACTGTAAA
colexBWT($\mathcal{M}$)	AAAGGCGG$$$TTACTGTAAA

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

A$#	G
A$. . .	G
A$. . .	C

U = A$

concBWT

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

U = A$

mdolBWT

$\texttt{A\$}_1\texttt{ATC}$	C
$\texttt{A\$}_2\texttt{GG}$	G
$\texttt{A\$}_3\texttt{TG}$	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$$$TTACTGTAAA	lexicographic order of strings
multidolBWT($\mathcal{M}$)	bwt($T_1\$_1 T_2\$_2 \cdots T_k\$_k$) GAGAAGCG$$$TTATCTGAAA	input order of strings
concatBWT($\mathcal{M}$)	bwt($T_1\$ T_2\$ \cdots T_k\$\#$) AAGAGGGC$$$TTACTGTAAA	lexicographic order of subsequent strings in input
colexBWT($\mathcal{M}$)	multidol($\mathcal{M}, \gamma$), γ = colex AAAGGCGG$$$TTACTGTAAA	colexicographic order

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$$$TTACTGTAAA	lexicographic order of strings
multidolBWT($\mathcal{M}$)	bwt($T_1\$_1 T_2\$_2 \cdots T_k\$_k$) GAGAAGCG$$$TTATCTGAAA	input order of strings
concatBWT($\mathcal{M}$)	bwt($T_1\$T_2\$ \cdots T_k\$\#$) AAGAGGGC$$$TTACTGTAAA	lexicographic order of subsequent strings in input
colexBWT($\mathcal{M}$)	multidol($\mathcal{M}, \gamma$), γ = colex AAAGGCGG$$$TTACTGTAAA	colexicographic order

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

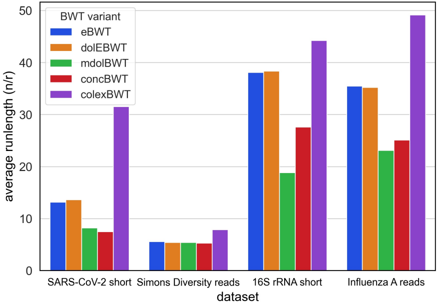
$\mathcal{M}_2$ = [ACG,ATATG,GGA,TGA,ATCA] mdolBWT($\mathcal{M}_2$) = GGAAAGGC$$$TTACTGTAAA

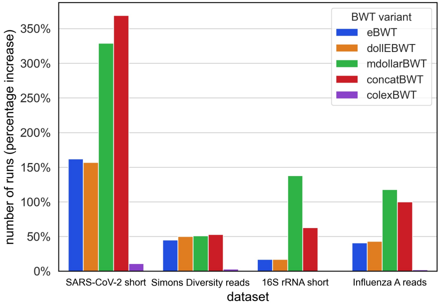
$\mathcal{M}_1$ = [ATATG,TGA,ACG,ATCA,GGA] concBWT($\mathcal{M}_1$) = AAGAGGGC$$$TTACTGTAAA

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

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) (up to a factor of 4.2 in our experiments)

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) (up to a factor of 4.2 in our experiments)

We suggest

- to standardize the definition of r (colexBWT or optBWT)
- optBWT now implemented (see Cenzato and L., WCTA 2022; Cenzato, Guerrini, L., Rosone, forthcoming)

4. A side question

What is the output order of the concatBWT?

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

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?

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

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?

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

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

BWT(beacd#) = de#acb ⇝ deacb

What is the output order of the concatBWT?

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

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

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

output = BWT(**input#**)

What is the output order of the concatBWT?

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

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

BWT(beacd#) = de#acb ⇝ deacb

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

Part III:

Conclusion

Dollar or no dollar, that is the question.

Conclusion

The two definitions of the BWT (with and without dollar) are non-equivalent. In particular,

Conclusion

The two definitions of the BWT (with and without dollar) are non-equivalent. In particular,

1. differences in the transform itself: $r(T)$ vs. $r(T\$)$

Conclusion

The two definitions of the BWT (with and without dollar) are non-equivalent. In particular,

1. differences in the transform itself: $r(T)$ vs. $r(T\$)$
2. BWT construction: cannot use SA when no dollar is present

Conclusion

The two definitions of the BWT (with and without dollar) are non-equivalent. In particular,

1. differences in the transform itself: $r(T)$ vs. $r(T\$)$
2. BWT construction: cannot use SA when no dollar is present
3. BWT of string collections: several non-equivalent methods in use

Acknowledgements (co-authors)

Marinella Sciortino
(Univ. of Palermo)

Shunsuke Inenanaga
(Kyushu Univ.)

Christina Boucher
(Univ. of Florida)

Massimiliano Rossi
(Illumina Inc.)

Sara Giuliani
(Univ. of Verona)

Davide Cenzato
(Univ. of Verona)

Francesco Masillo
(Univ. of Verona)

Literature

- C. Boucher, D. Cenzato, Zs. Lipták, M. Rossi, M. Sciortino: Computing the original eBWT faster, simpler, and with less memory. SPIRE 2021.
- S. Giuliani, S. Inenaga, Zs. Lipták, M. Sciortino: On bit catastrophes for the Burrows-Wheeler-Transform, forthcoming.
- D. Cenzato and Zs. Lipták: A theoretical and experimental analysis of BWT variants for string collections, CPM 2022.
- D. Cenzato and Zs. Lipták: Computing the optimal BWT using SAIS, WCTA 2022.
- D. Cenzato, V. Guerrini, Zs. Lipták, and G. Rosone: Computing the optimal BWT for very large string collections, submitted.

Thank you for your attention!

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