BWT everywhere

Zsuzsanna Lipták

University of Verona (Italy)

CPM 2024
Fukuoka, June 26, 2024

The BWT

BWT
RACE TAXI
GRAN TURISMO
DEKRA
AUTOHERO
DTM
HAUPT RACING TEAM

The BWT

BWT
RACE TAXI
GRAN TURISMO
AUTOHERO
DTM
DEKRA

ASTON MA
PERONI
BWT
crypto.com
BOMBARDIER
JCB
TTEL

The BWT

(Here BWT stands for: Best Water Technology)

The Burrows-Wheeler-Transform

T = `fukuoka`. The BWT is a permutation of T: $\texttt{bwt}(T) = \texttt{kaouufk}$

The Burrows-Wheeler-Transform

T = `fukuoka`. The BWT is a permutation of T: bwt(T) = `kaouufk`

all rotations (conjugates)

```
fukuoka
ukuokaf
kuokafu
uokafuk
okafuku
kafukuo
afukuok
```

The Burrows-Wheeler-Transform

T = fukuoka. The BWT is a permutation of T: bwt(T) = kaouufk

all rotations (conjugates)

fukuoka
ukuokaf
kuokafu
uokafuk
okafuku
kafukuo
afukuok

$\longrightarrow$
lexicographic
order

The Burrows-Wheeler-Transform

T = `fukuoka`. The BWT is a permutation of T: $\mathrm{bwt}(T)$ = `kaouufk`

all rotations (conjugates)		*all rotations, sorted*
		L
`fukuoka`		`afukuok`
`ukuokaf`	$\longrightarrow$ lexicographic order	`fukuoka`
`kuokafu`		`kafukuo`
`uokafuk`		`kuokafu`
`okafuku`		`okafuku`
`kafukuo`		`ukuokaf`
`afukuok`		`uokafuk`

The Burrows-Wheeler-Transform

$T = \texttt{fukuoka}$. The BWT is a permutation of T: $\mathrm{bwt}(T) = \texttt{kaouufk}$

all rotations (conjugates)

```
fukuoka
ukuokaf
kuokafu
uokafuk
okafuku
kafukuo
afukuok
```

$\longrightarrow$
lexicographic
order

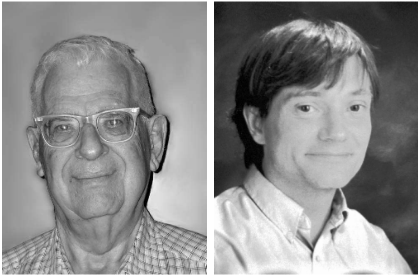
all rotations, sorted

```
      L
afukuok
fukuoka
kafukuo
kuokafu
okafuku
ukuokaf
uokafuk
```

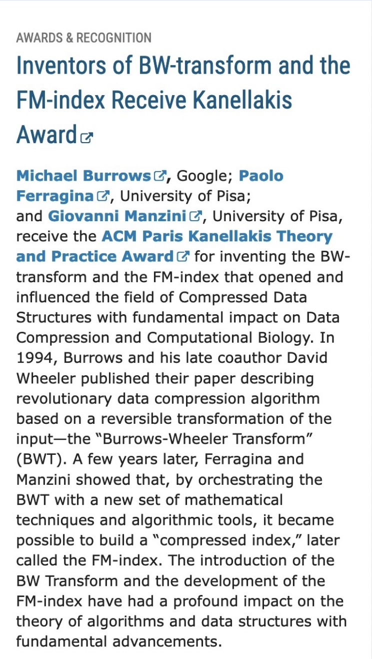
$\mathrm{BWT}(T) =$ concatenation of last characters $= L$

The Burrows-Wheeler Transform

- introduced by Burrows and Wheeler in 1994
- a reversible string transform
- basis of a highly effective lossless text compression algorithm
- basis of compressed data structures (compressed text indexes)



source: Adjeroh, Bell, Mukerjee (2008)

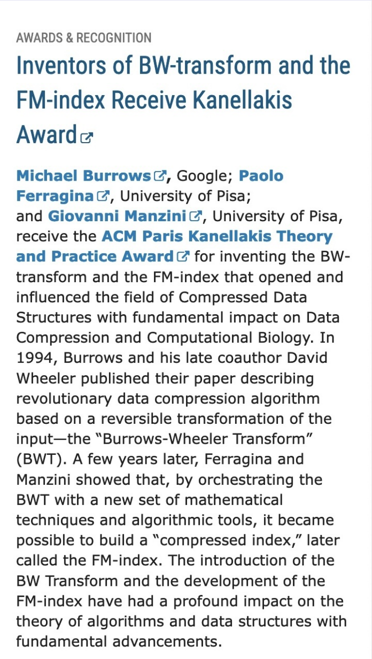


AWARDS & RECOGNITION

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.

- 2022 ACM Kanellakis Theory and Practice Award
- for BWT and FM-index (Ferragina & Manzini 2000, 2005)

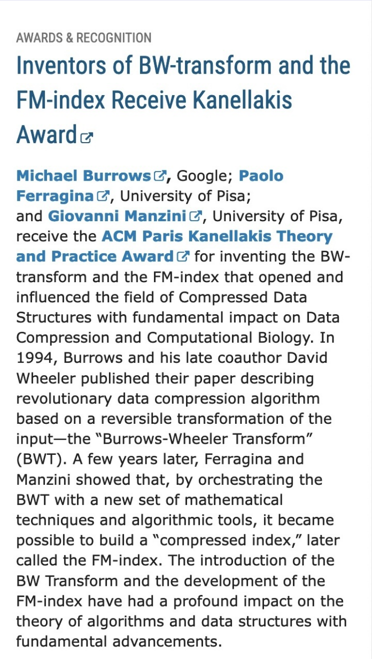


AWARDS & RECOGNITION

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.

- 2022 ACM Kanellakis Theory and Practice Award
- for BWT and FM-index (Ferragina & Manzini 2000, 2005)
- *"...that opened and influenced the field of Compressed Data Structures with fundamental impact on Data Compression and Computational Biology"*



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.

- 2022 ACM Kanellakis Theory and Practice Award
- for BWT and FM-index (Ferragina & Manzini 2000, 2005)
- *". . . that opened and influenced the field of Compressed Data Structures with fundamental impact on Data Compression and Computational Biology"*
- some bioinformatics tools:
  - `bwa, bwa-sw, bwa-mem` (Li & Durbin, 2009, 2010, Li 2013) $>$ 55,000 cit.
  - `bowtie, bowtie2` (Langmead et al., 2009, 2012) $>$ 70,000 cit.

This talk is about other uses of the BWT.

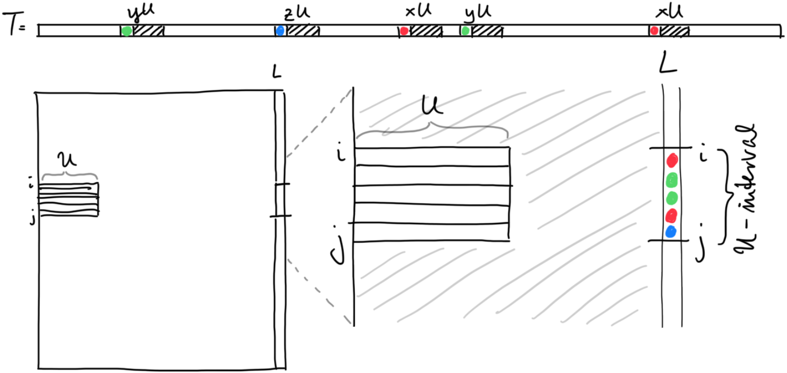
This talk is about other uses of the BWT.

1. distance measures based on the BWT
2. generating random de Bruijn sequences with the BWT
3. analyzing different BWT variants for string collections
4. why a common method for BWT of text collections is not a good idea

Our tools for this talk

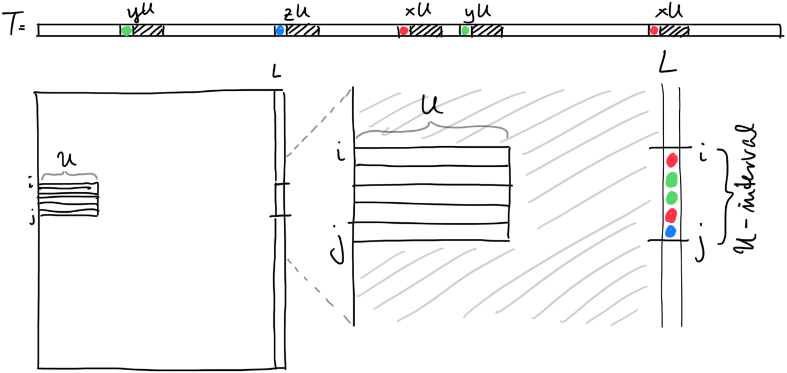
Tool 1: *U*-intervals

Def. Let U be a substring of T. The U-interval of $L = \text{bwt}(T)$ is $[i, j]$, where the conjugates in positions $k \in [i, j]$ are exactly those starting with U:



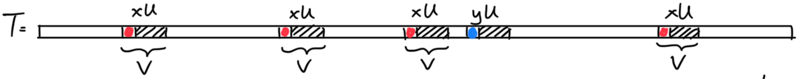
Tool 1: U-intervals

Def. Let U be a substring of T. The U-interval of $L = \text{bwt}(T)$ is $[i, j]$, where the conjugates in positions $k \in [i, j]$ are exactly those starting with U:

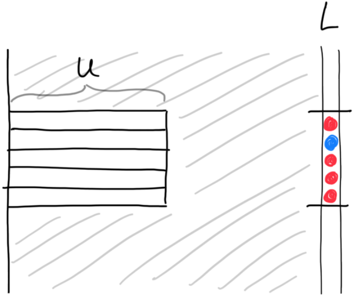


N.B.: $L[i..j]$ = left-context of U; $[i, j] \cong$ SA-interval of U (here: CA)

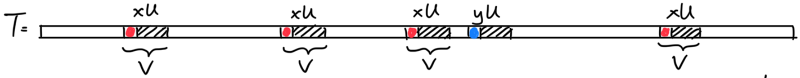
Why is the BWT so good in compression?



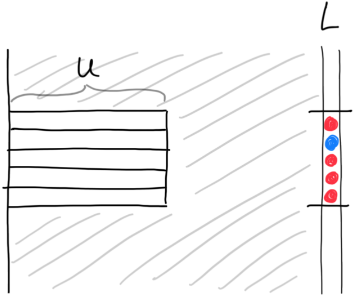
many occurrences of $V = xu$ ⇒ many x's in u-interval



Why is the BWT so good in compression?

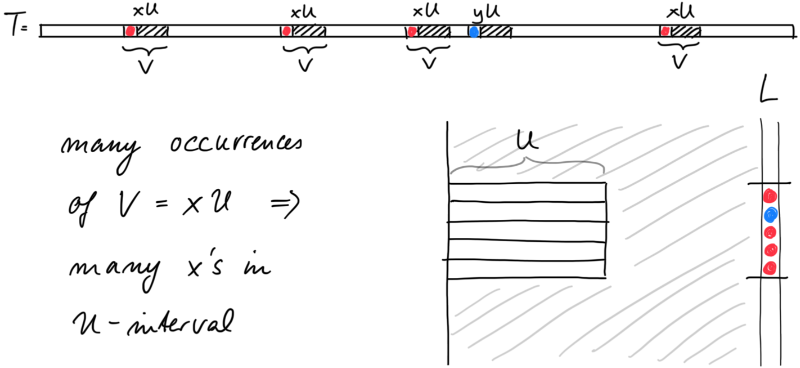


many occurrences of $V = xU$ ⇒ many x's in U-interval



- T has many repeated substrings ⇒ many U-intervals mostly same character
- $L = \text{bwt}(T)$ has few runs ⇒ runlength encoding (RLE) is good

Why is the BWT so good in compression?



many occurrences of $V = xU$ ⇒ many x's in U-interval

- T has many repeated substrings ⇒ many U-intervals mostly same character
- $L = \text{bwt}(T)$ has few runs ⇒ runlength encoding (RLE) is good

$\texttt{bbbacccccccccccccccccccaaaaa} \mapsto \texttt{b}^3\texttt{a}^1\texttt{c}^{18}\texttt{a}^5$

Tool 2: The extended BWT

(Mantaci, Restivo, Rosone, Sciortino, TCS, 2007)

Ex. $\mathcal{M} = \{\texttt{fu}, \texttt{k}, \texttt{uoka}\}$. The eBWT is a permutation of the characters of $\mathcal{M}$: eBWT$(\mathcal{M}) = \texttt{kuokufa}$.

all rotations (conjugates)

fu
uf
k
uoka
okau
kauo
auok

$\xrightarrow[\text{omega order}]{}$

all rotations, sorted

auok	k
fu	u
kauo	o
k	k
okau	u
uf	f
uoka	a

N.B. $\texttt{kauo} <_\omega \texttt{k}$: $\quad \texttt{kauo} \cdot \texttt{kauo} \cdots <_{\text{lex}} \texttt{k} \cdot \texttt{k} \cdot \texttt{k} \cdot \texttt{k} \cdots$

The extended BWT (cont.)

Def. (omega-order): $T <_\omega S$ if (a) $T^\omega <_{\text{lex}} S^\omega$, or
(b) $T^\omega = S^\omega,\ T = U^k, S = U^m$ and $k < m$

$\mathcal{M} = \{\texttt{fu}, \texttt{k}, \texttt{uoka}\}$

lex-order		*omega-order*	
auok	k	auok	k
fu	u	fu	u
k	k	kauo	o
kauo	o	k	k
okau	u	okau	u
uf	f	uf	f
uoka	a	uoka	a

(**N.B.** With the lex-order, the LF-property would not hold.)

The extended BWT (cont.)

- omega-order instead of lex-order
- the eBWT inherits BWT properties: clustering effect, reversibility, useful for lossless text compression, efficient pattern matching, . . .
- However, until recently no linear-time algorithm was known.

The extended BWT (cont.)

- omega-order instead of lex-order
- the eBWT inherits BWT properties: clustering effect, reversibility, useful for lossless text compression, efficient pattern matching, . . .
- However, until recently no linear-time algorithm was known.

Since 2021: linear-time algorithms and implementations available

- First linear-time algorithm
  (Bannai, Kärkkäinen, Köppl, Piatkowski, CPM 2021)
- We significantly simplified this algorithm
  (Boucher, Cenzato, L., Rossi, Sciortino, SPIRE 2021)
- . . . and gave efficient implementations of the eBWT (`cais`,`pfpebwt` 2021)
- Later we gave an *r*-index based on the eBWT (—, Inf. & Comp., 2024)

Tool 3: The standard permutation

Def. Given a string V, its standard permutation π_V is defined by: $\pi_V(i) < \pi_V(j)$ if (i) $V_i < V_j$, or (ii) $V_i = V_j$ and $i < j$.

In other words, π_V is a stable sort of the characters of V.

Example: $V =$ `kaouufk`

0	1	2	3	4	5	6
k	a	o	u	u	f	k
a	f	k	k	o	u	u
0	1	2	3	4	5	6

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

(If V is a BWT, then π_V is called LF-mapping.)

The standard permutation (cont.)

- If V is a BWT, then π_V is called LF-mapping.

The standard permutation (cont.)

- If V is a BWT, then π_V is called LF-mapping.
- With π_V we can recover (a conjugate of) T from bwt(T) back-to-front:

  Ex. $V = \texttt{kaouufk}$, $\pi_V = (0, 2, 4, 6, 3, 5, 1)$

The standard permutation (cont.)

- If V is a BWT, then π_V is called LF-mapping.
- With π_V we can recover (a conjugate of) T from bwt(T) back-to-front:

  Ex. $V =$ `kaouufk`, $\pi_V = (0, 2, 4, 6, 3, 5, 1)$ `afukuok`

The standard permutation (cont.)

- If V is a BWT, then π_V is called LF-mapping.
- With π_V we can recover (a conjugate of) T from bwt(T) back-to-front:

  Ex. $V =$ `kaouufk`, $\pi_V = (0, 2, 4, 6, 3, 5, 1)$ `afukuok`
  (or given pos. 1: `fukuoka`)

The standard permutation (cont.)

- If V is a BWT, then π_V is called LF-mapping.
- With π_V we can recover (a conjugate of) T from bwt(T) back-to-front:

  Ex. $V =$ `kaouufk`, $\pi_V = (0, 2, 4, 6, 3, 5, 1)$ `afukuok`
  (or given pos. 1: `fukuoka`)
- Similarly, we can recover (conjugates of) $\mathcal{M}$ from eBWT($\mathcal{M}$):

  Ex. $V =$ `kuokufa`, $\pi_V = (0, 2, 4, 6)(1, 5)(3)$

The standard permutation (cont.)

- If V is a BWT, then π_V is called LF-mapping.
- With π_V we can recover (a conjugate of) T from bwt(T) back-to-front:

  Ex. $V = \texttt{kaouufk}$, $\pi_V = (0, 2, 4, 6, 3, 5, 1)$ `afukuok`
  (or given pos. 1: `fukuoka`)
- Similarly, we can recover (conjugates of) $\mathcal{M}$ from eBWT($\mathcal{M}$):

  Ex. $V = \texttt{kuokufa}$, $\pi_V = (0, 2, 4, 6)(1, 5)(3)$ `auok, fu, k`

The standard permutation (cont.)

- If V is a BWT, then π_V is called LF-mapping.
- With π_V we can recover (a conjugate of) T from bwt(T) back-to-front:

  Ex. $V =$ `kaouufk`, $\pi_V = (0,2,4,6,3,5,1)$ `afukuok`
  (or given pos. 1: `fukuoka`)
- Similarly, we can recover (conjugates of) $\mathcal{M}$ from eBWT($\mathcal{M}$):

  Ex. $V =$ `kuokufa`, $\pi_V = (0,2,4,6)(1,5)(3)$ `auok, fu, k`
  (or given the positions: `uoka, fu, k`)

The standard permutation (cont.)

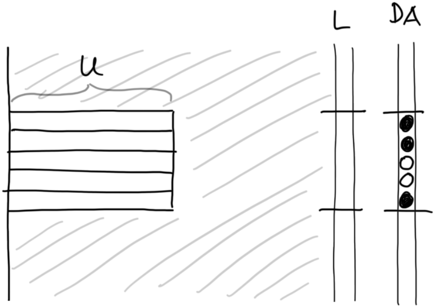
- If V is a BWT, then π_V is called LF-mapping.
- With π_V we can recover (a conjugate of) T from bwt(T) back-to-front:

  Ex. $V =$ `kaouufk`, $\pi_V = (0, 2, 4, 6, 3, 5, 1)$ `afukuok`
  (or given pos. 1: `fukuoka`)
- Similarly, we can recover (conjugates of) $\mathcal{M}$ from eBWT($\mathcal{M}$):

  Ex. $V =$ `kuokufa`, $\pi_V = (0, 2, 4, 6)(1, 5)(3)$ `auok, fu, k`
  (or given the positions: `uoka, fu, k`)

Thm. (Folklore) A string V is the BWT of a primitive string if and only if π_V is cyclic.

Distance / similarity measures



Mantaci, Restivo, Rosone, Sciortino, ToCS 2007

Distance/similarity based on eBWT

Idea: Conjugates of similar strings should mix well in the eBWT.

Ex.: $S =$ `kyoto`, $T =$ `tokyo`.

conjugates	L	DA (document array)
kyoto	o	S
kyoto	o	T
okyot	t	S
okyot	t	T
otoky	y	S
otoky	y	T
tokyo	o	S
tokyo	o	T
yotok	k	S
yotok	k	T

runlengths of DA: $i_0, i_1, \ldots, i_\ell$

Def. (delta-distance)
$\delta(S, T) = \sum_{j=0}^{\ell}(i_j - 1)$

$\delta(\texttt{tokyo}, \texttt{kyoto}) = 0$

$S = \texttt{fukuoka}$,
$T = \texttt{fujioka}$.

conjugates	L	DA
afujiok	k	T
afukuok	k	S
fujioka	a	T
fukuoka	a	S
iokafuj	j	T
jiokafu	u	T
kafujio	o	T
kafukuo	o	S
kuokafu	u	S
okafuji	i	T
ujiokaf	f	T
ukuokaf	f	S
uokafuk	k	S

Def. (delta-distance)
$\delta(S,T) = \sum_{j=0}^{\ell}(i_j - 1)$

$$DA = T^1 S^1 T^1 S^1 T^3 S^2 T^2 S^2$$

$$\delta(S,T) = 2 + 1 + 1 + 1 = 5$$

$S =$ fukuoka,
$T =$ fujioka.

conjugates	L	DA
afujiok	k	T
afukuok	k	S
fujioka	a	T
fukuoka	a	S
iokafuj	j	T
jiokafu	u	T
kafujio	o	T
kafukuo	o	S
kuokafu	u	S
okafuji	i	T
ujiokaf	f	T
ukuokaf	f	S
uokafuk	k	S

Def. (delta-distance)
$\delta(S,T) = \sum_{j=0}^{\ell}(i_j - 1)$

$DA = T^1 S^1 T^1 S^1 T^3 S^2 T^2 S^2$

$\delta(S,T) = 2 + 1 + 1 + 1 = 5$

- δ has been used in bioinformatics, malware analysis, artwork comparison, ...
- a modification called 'BW similarity distribution' uses the expectation of the i_j and the Shannon-entropy (Yang et al. 2010, Yang et al. 2010, Louza et al. 2019)

Let $P_1 \cdot P_2 \cdots P_m$ a parsing $\mathcal{P}$ of DA.

$S =$ fukuoka,
$T =$ fujioka.

conjugates	L	DA
afujiok	k	T
afukuok	k	S
fujioka	a	T
fukuoka	a	S
iokafuj	j	T
jiokafu	u	T
kafujio	o	T
kafukuo	o	S
kuokafu	u	S
okafuji	i	T
ujiokaf	f	T
ukuokaf	f	S
uokafuk	k	S

Def. $dist_{\mathcal{P}}(S,T) = \sum_{i=1}^{m} ||P_i|_S - |P_i|_T|$

where $|P_i|_x$ is the multiplicity of x in P_i

Ex. Let $\mathcal{P}$ be the parsing
DA $= (TS)(TS)(T)(T)(TS)(S)(T)(T)(S)(S)$,
then $dist_{\mathcal{P}}(S,T) = 7$.

$S =$ `fukuoka`,
$T =$ `fujioka`.

conjugates	L	DA
afujiok	k	T
afukuok	k	S
fujioka	a	T
fukuoka	a	S
iokafuj	j	T
jiokafu	u	T
kafujio	o	T
kafukuo	o	S
kuokafu	u	S
okafuji	i	T
ujiokaf	f	T
ukuokaf	f	S
uokafuk	k	S

Let $P_1 \cdot P_2 \cdots P_m$ a parsing $\mathcal{P}$ of DA.

Def. $dist_{\mathcal{P}}(S,T) = \sum_{i=1}^{m} ||P_i|_S - |P_i|_T|$

where $|P_i|_x$ is the multiplicity of x in P_i

Ex. Let $\mathcal{P}$ be the parsing
$\mathrm{DA} = (TS)(TS)(T)(T)(TS)(S)(T)(T)(S)(S)$,
then $dist_{\mathcal{P}}(S,T) = 7$.

This can be used e.g. to simulate the k-mer distance
(aka q-gram distance, Ukkonen 1992):

Def. (k-mer distance)
$dist_k(S,T) =$
$\sum_{|U|=k} |mult(S,U) - mult(T,U)|$

$S =$ fukuoka,
$T =$ fujioka.

conjugates	L	DA
afujiok	k	T
afukuok	k	S
fujioka	a	T
fukuoka	a	S
iokafuj	j	T
jiokafu	u	T
kafujio	o	T
kafukuo	o	S
kuokafu	u	S
okafuji	i	T
ujiokaf	f	T
ukuokaf	f	S
uokafuk	k	S

Let $P_1 \cdot P_2 \cdots P_m$ a parsing $\mathcal{P}$ of DA.

Def. $dist_{\mathcal{P}}(S,T) = \sum_{i=1}^{m} ||P_i|_S - |P_i|_T|$

where $|P_i|_x$ is the multiplicity of x in P_i

Ex. Let $\mathcal{P}$ be the parsing
$\text{DA} = (TS)(TS)(T)(T)(TS)(S)(T)(T)(S)(S)$,
then $dist_{\mathcal{P}}(S,T) = 7$.

This can be used e.g. to simulate the k-mer distance
(aka q-gram distance, Ukkonen 1992):

Def. (k-mer distance)
$dist_k(S,T) =$
$\sum_{|U|=k} |mult(S,U) - mult(T,U)|$

$dist_2(S,T) = 7$

$S = \texttt{fukuoka}$,
$T = \texttt{fujioka}$.

conjugates	L	DA
afujiok	k	T
afukuok	k	S
fujioka	a	T
fukuoka	a	S
iokafuj	j	T
jiokafu	u	T
kafujio	o	T
kafukuo	o	S
kuokafu	u	S
okafuji	i	T
ujiokaf	f	T
ukuokaf	f	S
uokafuk	k	S

Let $L = eBWT(S, T)$, and $DA = P_1 \cdots P_r$ the parsing of the DA where P_i corresponds to the ith run of L.

Def. (rho: monotonic block parsing)
$\rho(S, T) = \sum_{i=1}^{r} ||P_i|_S - |P_i|_T|$

$S =$ fukuoka,
$T =$ fujioka.

conjugates	L	DA
afujiok	k	T
afukuok	k	S
fujioka	a	T
fukuoka	a	S
iokafuj	j	T
jiokafu	u	T
kafujio	o	T
kafukuo	o	S
kuokafu	u	S
okafuji	i	T
ujiokaf	f	T
ukuokaf	f	S
uokafuk	k	S

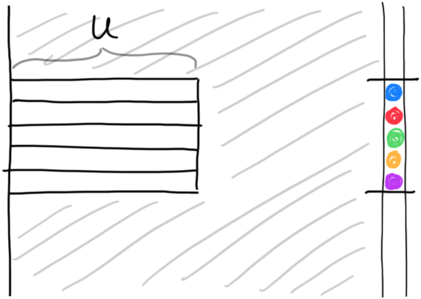
Let $L = eBWT(S, T)$, and $DA = P_1 \cdots P_r$ the parsing of the DA where P_i corresponds to the ith run of L.

Def. (rho: monotonic block parsing)
$\rho(S, T) = \sum_{i=1}^{r} ||P_i|_S - |P_i|_T|$

Ex.
$\mathrm{DA} = (TS)(TS)(T)(T)(TS)(S)(T)(TS)(S)$,

$\rho(S, T) = 5$

Generating random de Bruijn sequences



L. & Parmigiani, LATIN 2024

de Bruijn sequences

Def. A de Bruijn sequence (dB sequence) of order k over an alphabet Σ is a circular string in which every k-mer occurs exactly once as a substring.

k-mer $=$ string of length k

Ex. $k = 3$: $\underset{0}{\texttt{a}}\underset{1}{\texttt{a}}\underset{2}{\texttt{a}}\underset{3}{\texttt{b}}\underset{4}{\texttt{a}}\underset{5}{\texttt{b}}\underset{6}{\texttt{b}}\underset{7}{\texttt{b}}$ (binary)

de Bruijn sequences

Def. A de Bruijn sequence (dB sequence) of order k over an alphabet Σ is a circular string in which every k-mer occurs exactly once as a substring.

k-mer = string of length k

Ex. $k = 3$: $\begin{array}{cccccccc} \texttt{a} & \texttt{a} & \texttt{a} & \texttt{b} & \texttt{a} & \texttt{b} & \texttt{b} & \texttt{b} \\ 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array}$ (binary)

k-mer	position
aaa	0
aab	1
aba	2
abb	4
baa	7
bab	3
bba	6
bbb	5

de Bruijn sequences

Def. A de Bruijn sequence (dB sequence) of order k over an alphabet Σ is a circular string in which every k-mer occurs exactly once as a substring.

k-mer = string of length k

Ex. $k = 3$: $\underset{0\,1\,2\,3\,4\,5\,6\,7}{\texttt{aaababbb}}$ (binary)

$k = 3$: aaacaabbabcacccabacbccbbbcb (ternary)

k-mer	position
aaa	0
aab	1
aba	2
abb	4
baa	7
bab	3
bba	6
bbb	5

de Bruijn sequences

Def. A de Bruijn sequence (dB sequence) of order k over an alphabet Σ is a circular string in which every k-mer occurs exactly once as a substring.

k-mer = string of length k

Ex. $k = 3$: $\underset{0\,1\,2\,3\,4\,5\,6\,7}{\texttt{aaababbb}}$ (binary)

k-mer	position
aaa	0
aab	1
aba	2
abb	4
baa	7
bab	3
bba	6
bbb	5

$k = 3$: `aaacaabbabcacccabacbccbbbcb` (ternary)

Easy: length of a dB sequence is σ^k ($\sigma = |\Sigma|$)

de Bruijn sequences

- de Bruijn sequences exist for every k and σ

de Bruijn sequences

- de Bruijn sequences exist for every k and σ
- There are $(\sigma!)^{\sigma^{k-1}}/\sigma^k$ dB sequences of order k

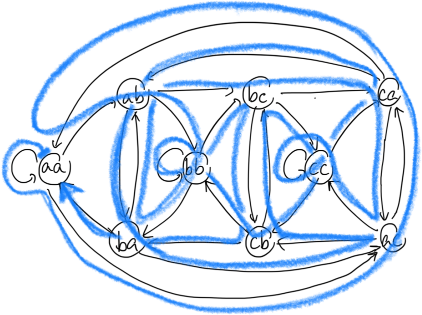
(Fly Sainte-Marie 1894,
Tatyana van Aardenne-Ehrenfest and Nicolaas de Bruijn 1951: BEST Thm.)

de Bruijn sequences

- de Bruijn sequences exist for every k and σ
- There are $(\sigma!)^{\sigma^{k-1}}/\sigma^k$ dB sequences of order k

(Fly Sainte-Marie 1894, Tatyana van Aardenne-Ehrenfest and Nicolaas de Bruijn 1951: BEST Thm.)

- dB sequences correspond to Euler cycles in the dB graph



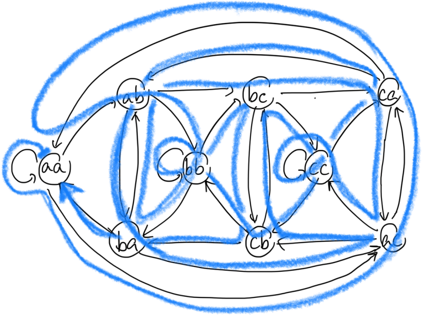
aaacaabbabcacccabacbccbbbcb

de Bruijn sequences

- de Bruijn sequences exist for every k and σ
- There are $(\sigma!)^{\sigma^{k-1}}/\sigma^k$ dB sequences of order k

(Fly Sainte-Marie 1894,
Tatyana van Aardenne-Ehrenfest and Nicolaas de Bruijn 1951: BEST Thm.)

- dB sequences correspond to Euler cycles in the dB graph



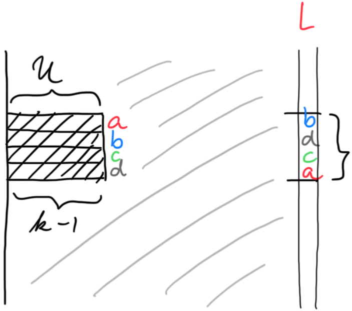
`aaacaabbabcacccabacbccbbbcb`

(one of the 373 248 dB seqs for $\sigma = 3, k = 3$)

Applications of de Bruijn sequences

- pseudo-random bit generators
- experimental design: reaction time experiments, imaging studies (MRI)
- computational biology: DNA probe design, DNA microarray, DNA synthesis
- cryptographic protocols
- ...

The BWT of de Bruijn sequences

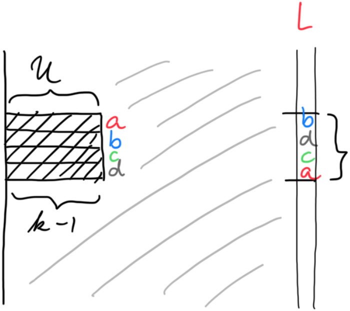


$u \in \Sigma^{k-1}$ occurs σ times

u-interval contains a permutation of Σ

(in particular, BWT+RLE does not compress well: many runs!)

The BWT of de Bruijn sequences



$u \in \Sigma^{k-1}$ occurs 6 times

u-interval contains a permutation of Σ

(in particular, BWT+RLE does not compress well: many runs!)

N.B. From now on: binary dB sequences (for simplicity).

Construction algorithms

Many algorithms for constructing dB sequences:

- H. Fredricksen: *A survey of full length nonlinear shift register cycle algorithms*, 1982 (classic survey)
- Gabric & Sawada, Discr. Math. 2022
- website `debruijnsequence.org` run by Joe Sawada and others

Construction algorithms

Many algorithms for constructing dB sequences:

- H. Fredricksen: *A survey of full length nonlinear shift register cycle algorithms*, 1982 (classic survey)
- Gabric & Sawada, Discr. Math. 2022
- website `debruijnsequence.org` run by Joe Sawada and others

Most construct:

- one particular dB sequence (e.g. the lex-least dB sequence), or

Construction algorithms

Many algorithms for constructing dB sequences:

- H. Fredricksen: *A survey of full length nonlinear shift register cycle algorithms*, 1982 (classic survey)
- Gabric & Sawada, Discr. Math. 2022
- website `debruijnsequence.org` run by Joe Sawada and others

Most construct:

- one particular dB sequence (e.g. the lex-least dB sequence), or
- a small subset of dB sequences (e.g. linear feedback shift registers)

Construction algorithms

Many algorithms for constructing dB sequences:

- H. Fredricksen: *A survey of full length nonlinear shift register cycle algorithms*, 1982 (classic survey)
- Gabric & Sawada, Discr. Math. 2022
- website `debruijnsequence.org` run by Joe Sawada and others

Most construct:

- one particular dB sequence (e.g. the lex-least dB sequence), or
- a small subset of dB sequences (e.g. linear feedback shift registers)

k	4	5	6	7	10	15	20
#LFSRs	2	6	6	18	60	1 800	24 000
#dBseqs	16	2048	67 108 864	$1.44 \cdot 10^{17}$	$1.3 \cdot 10^{151}$	$3.63 \cdot 10^{4927}$	$2.47 \cdot 10^{157820}$

- number of binary dB sequences $= 2^{2^{k-1}-k}$

Construction of random dB sequences

- The only algorithms able to construct any dB sequence are based on finding Eulerian cycles in de Bruijn graphs (Hierholzer, Fleury)

Construction of random dB sequences

- The only algorithms able to construct any dB sequence are based on finding Eulerian cycles in de Bruijn graphs (Hierholzer, Fleury)
- Surprisingly, no practical algorithms for random dB sequence construction that can output any dB sequence with positive probability.

Construction of random dB sequences

- The only algorithms able to construct any dB sequence are based on finding Eulerian cycles in de Bruijn graphs (Hierholzer, Fleury)
- Surprisingly, no practical algorithms for random dB sequence construction that can output any dB sequence with positive probability.
- Our algorithm does just that!

Construction of random dB sequences

- The only algorithms able to construct any dB sequence are based on finding Eulerian cycles in de Bruijn graphs (Hierholzer, Fleury)
- Surprisingly, no practical algorithms for random dB sequence construction that can output any dB sequence with positive probability.
- Our algorithm does just that!
- . . . in near-linear time $\mathcal{O}(n\alpha(n))$, $n =$ length of dB sequence
  $\alpha =$ inverse Ackermann function

Construction of random dB sequences

- The only algorithms able to construct any dB sequence are based on finding Eulerian cycles in de Bruijn graphs (Hierholzer, Fleury)
- Surprisingly, no practical algorithms for random dB sequence construction that can output any dB sequence with positive probability.
- Our algorithm does just that!
- . . . in near-linear time $\mathcal{O}(n\alpha(n))$, $n =$ length of dB sequence
  $\alpha =$ inverse Ackermann function
- . . . and it is beautifully simple at that!

The BWT of a dB sequence

$T = \texttt{aaababbb}, k = 3$

a	a	a	b	a	b	b	b
a	a	b	a	b	b	b	a
a	b	a	b	b	b	a	a
a	b	b	b	a	a	a	b
b	a	a	a	b	a	b	b
b	a	b	b	b	a	a	a
b	b	a	a	a	b	a	b
b	b	b	a	a	a	b	a

$\text{bwt}(\texttt{aaababbb}) = \texttt{baabbaba}$

The BWT of a dB sequence

$T = \texttt{aaababbb}, k = 3$

a	a	a	b	a	b	b	b
a	a	b	a	b	b	b	a
a	b	a	b	b	b	a	a
a	b	b	b	a	a	a	b
b	a	a	a	b	a	b	b
b	a	b	b	b	a	a	a
b	b	a	a	a	b	a	b
b	b	b	a	a	a	b	a

$\text{bwt}(\texttt{aaababbb}) = \texttt{baabbaba}$

$\text{bwt}(T) \in \{\texttt{ab}, \texttt{ba}\}^{2^{k-1}}$

The BWT of a dB sequence

Q. Is every string $V \in \{\texttt{ab},\texttt{ba}\}^{2^{k-1}}$ the BWT of a dB sequence?

The BWT of a dB sequence

Q. Is every string $V \in \{\texttt{ab},\texttt{ba}\}^{2^{k-1}}$ the BWT of a dB sequence?

A. No! e.g. $V = \texttt{abbababa}$, its standard permutation is

$$\pi_V = \left(\begin{smallmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 0 & 4 & 5 & 1 & 6 & 2 & 7 & 3 \end{smallmatrix}\right) = (0)(1,4,6,7,3)(2,5)$$

Indeed, $V = \text{eBWT}(\{\texttt{a}, \texttt{aabbb}, \texttt{ab}\})$.

The BWT of a dB sequence

Q. Is every string $V \in \{\texttt{ab},\texttt{ba}\}^{2^{k-1}}$ the BWT of a dB sequence?

A. No! e.g. $V = \texttt{abbababa}$, its standard permutation is

$$\pi_V = \left(\begin{smallmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 0 & 4 & 5 & 1 & 6 & 2 & 7 & 3 \end{smallmatrix}\right) = (0)(1,4,6,7,3)(2,5)$$

Indeed, $V = \text{eBWT}(\{\texttt{a}, \texttt{aabbb}, \texttt{ab}\})$.

Def. (Higgins, 2012) A binary de Bruijn set of order k is a multiset of total length 2^k such that every k-mer is the prefix of some rotation of some power of some string in $\mathcal{M}$.

Ex. $\mathcal{M} = \{\texttt{a},\texttt{ab},\texttt{aabbb}\}$ k-mers: $\texttt{aaa}, \texttt{aab}, \texttt{bab}, \ldots$

The basic theorem

Thm (Higgins, 2012) The set $\{\texttt{ab},\texttt{ba}\}^{2^{k-1}}$ is the set of eBWTs of binary de Bruijn sets of order k.

Corollary A string $V \in \{\texttt{ab},\texttt{ba}\}^{2^{k-1}}$ is the BWT of a dB sequence if and only if π_V is cyclic.

Our idea: Take a random $V \in \{\texttt{ab}, \texttt{ba}\}^{2^{k-1}}$ and turn it into the BWT of a dB sequence.

Lemma (Swap Lemma) Let V be a binary string, $V_i \neq V_{i+1}$, and V' the result of swapping V_i and V_{i+1}.

- If i and $i+1$ belong to distinct cycles in of π_V then the number of cycles decreases by one,
- otherwise it increases by one.

N.B.: a generalization of a technique from (Giuliani, L., Masillo, Rizzi, 2021)

Lemma (Swap Lemma) Let V be a binary string, $V_i \neq V_{i+1}$, and V' the result of swapping V_i and V_{i+1}.

- If i and $i+1$ belong to distinct cycles in of π_V then the number of cycles decreases by one,
- otherwise it increases by one.

N.B.: a generalization of a technique from (Giuliani, L., Masillo, Rizzi, 2021)

Ex. $V = \underset{0\,1\,2\,3\,4\,5\,6\,7}{\texttt{abbababa}}$, then $\pi_V = (0)(1,4,6,7,3)(2,5)$.

Lemma (Swap Lemma) Let V be a binary string, $V_i \neq V_{i+1}$, and V' the result of swapping V_i and V_{i+1}.

- If i and $i+1$ belong to distinct cycles in of π_V then the number of cycles decreases by one,
- otherwise it increases by one.

N.B.: a generalization of a technique from (Giuliani, L., Masillo, Rizzi, 2021)

Ex. $V = \underset{0\,1\,2\,3\,4\,5\,6\,7}{\texttt{abbababa}}$, then $\pi_V = (0)(1,4,6,7,3)(2,5)$.

- swap V_0 and V_1 : `babababa`, st. perm. $(0,4,6,7,3,1)(2,5)$

Lemma (Swap Lemma) Let V be a binary string, $V_i \neq V_{i+1}$, and V' the result of swapping V_i and V_{i+1}.

- If i and $i+1$ belong to distinct cycles in of π_V then the number of cycles decreases by one,
- otherwise it increases by one.

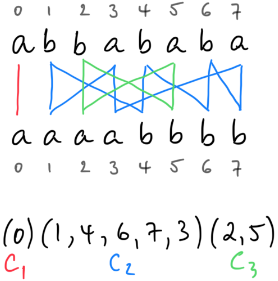
N.B.: a generalization of a technique from (Giuliani, L., Masillo, Rizzi, 2021)

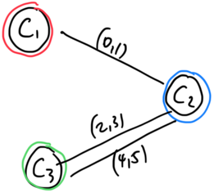
Ex. $V = \underset{01234567}{\texttt{abbababa}}$, then $\pi_V = (0)(1,4,6,7,3)(2,5)$.

- swap V_0 and V_1 : babababa, st. perm. $(0,4,6,7,3,1)(2,5)$
- swap V_2 and V_3 : baabbaba, st. perm. $(0,4,6,7,3,5,2,1)$

Invert baabbaba and output the dB seq $T =$ aaababbb.

How to choose the blocks to swap





- unhappy block: elements $2i, 2i+1$ are in different cycles
- cycle graph Γ_V: vertices = cycles, edges = unhappy blocks
- Spanning Trees of Γ_V = (BWTs of) dB sequences closest to V
- here 2 STs: BWTs of `aaabbbab`, `aaababbb`

BWT-based algorithm for generating random dB sequences

- first practical algorithm for constructing a random dB sequence which produces any dB sequence with positive probability
  - time $\mathcal{O}(n\alpha(n))$
  - space $\mathcal{O}(n)$

BWT-based algorithm for generating random dB sequences

- first practical algorithm for constructing a random dB sequence which produces any dB sequence with positive probability
  - time $\mathcal{O}(n\alpha(n))$
  - space $\mathcal{O}(n)$
- implementation: `github.com/lucaparmigiani/rnd_dbseq`
  - simple (less than 120 lines of C++ code)
  - fast (less than one second on a laptop for k up to 23)

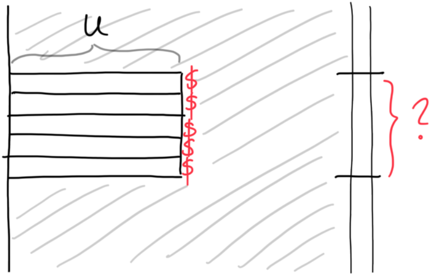
BWT-based algorithm for generating random dB sequences

- first practical algorithm for constructing a random dB sequence which produces any dB sequence with positive probability
  - time $\mathcal{O}(n\alpha(n))$
  - space $\mathcal{O}(n)$
- implementation: `github.com/lucaparmigiani/rnd_dbseq`
  - simple (less than 120 lines of C++ code)
  - fast (less than one second on a laptop for k up to 23)
- try it: `debruijnsequence.org/db/random`

BWT-based algorithm for generating random dB sequences

- first practical algorithm for constructing a random dB sequence which produces any dB sequence with positive probability
  - time $\mathcal{O}(n\alpha(n))$
  - space $\mathcal{O}(n)$
- implementation: `github.com/lucaparmigiani/rnd_dbseq`
  - simple (less than 120 lines of C++ code)
  - fast (less than one second on a laptop for k up to 23)
- try it: `debruijnsequence.org/db/random`
- can be straighforwardly extended to any constant-size alphabet (present on github)

On text indexes for string collections



Cenzato & L., CPM 2022, Bioinformatics 2024
Cenzato, Guerrini, L., Rosone, DCC 2023

BWT of string collections

All that glisters is not gold. (W. Shakespeare, The Merchant of Venice)

All that is referred to as extended BWT is **not** extended BWT.

BWT of string collections

- Often, **any** BWT of a string collection is called extended BWT.
- Many tools exist for BWT of string collections, but until 2021 none computed the original eBWT.

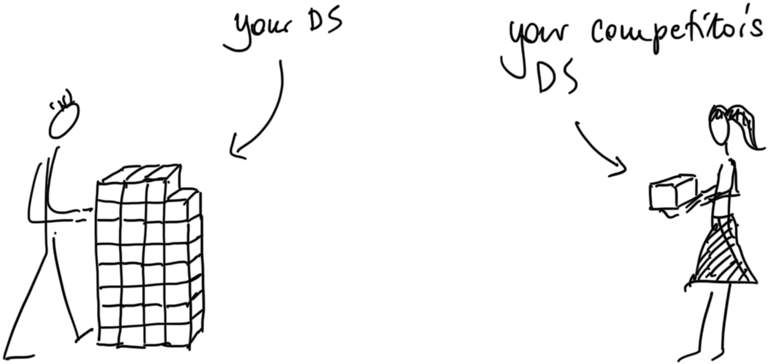
Q. So what do these tools compute?

The different BWT variants

(Cenzato & L., CPM 2022, Bioinformatics 2024)

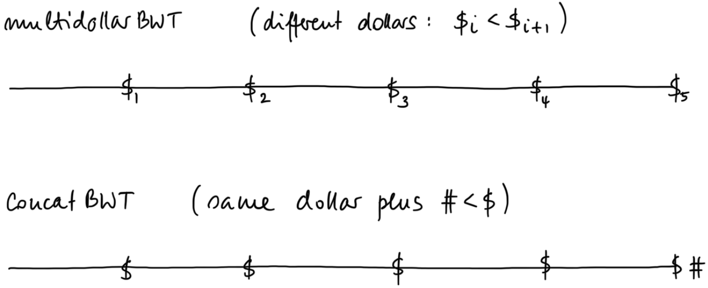
- We surveyed 18 different tools and the resulting BWT variants
- We identified 5 distinct BWT variants for string collections, . . .
- . . . and later added a 6th variant, the optimalBWT, which minimizes r (see later)
- All but the original eBWT use end-of-string symbols ($).
- The BWT variants differ also in the number of runs r.

size of data structures is $\mathcal{O}(r)$



BWT of text collections with dollars

- Most commonly, the strings are concatenated and then treated like one string.
- Two methods: multidollarBWT (and variations) and concatBWT


multidollarBWT (different dollars: $\$_i < \$_{i+1}$)
$\$_1$ $\$_2$ $\$_3$ $\$_4$ $\$_5$
ConcatBWT (same dollar plus $\# < \$$)
$ $ $ $ $ #

- We showed that all variants can be reduced to multidollarBWT.

Interesting intervals

Q. Where exactly do these BWT variants differ? **A.** in interesting intervals

Interesting intervals

Q. Where exactly do these BWT variants differ? **A.** in 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$AAA$`
colexBWT($\mathcal{M}$)	`AAAGGCGG$$$TTACTGT$AAA$`
concatBWT($\mathcal{M}$)	`AAGAGGGC$$$TTACTGT$AAA$`
optimalBWT	`AAAGGGGC$$$TTACTTG$AAA$`

in color: **interesting intervals**

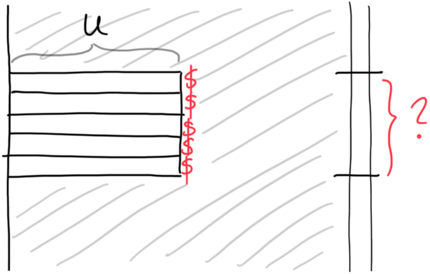
colex a.k.a. 'rlo'

Def. An interval $[i, j]$ is interesting if it is the $U\$$-interval of a left-maximal shared suffix U.

Def. An interval $[i, j]$ is interesting if it is the $U\$$-interval of a left-maximal shared suffix U.

Ex. $U = \mathtt{A}$
$\mathcal{M} = \{\mathtt{ATATG}, \mathtt{TG\underline{A}}, \mathtt{ACG}, \mathtt{ATC\underline{A}}, \mathtt{GG\underline{A}}\}$

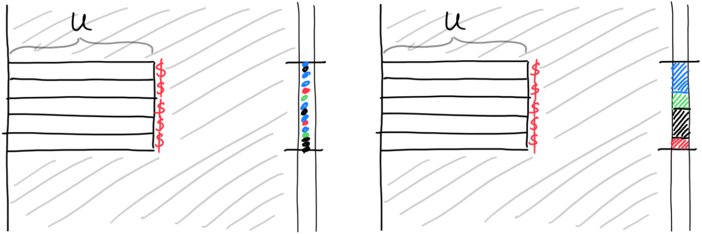
$\mathtt{A\$_2} \cdots$	G	$\mathtt{A\$_1} \cdots$	C
$\mathtt{A\$_4} \cdots$	C	$\mathtt{A\$_2} \cdots$	G
$\mathtt{A\$_5} \cdots$	G	$\mathtt{A\$_3} \cdots$	G
(input)		(colex)	



$U \in \Sigma^*$ is called a left-maximal shared suffix if there exist two strings $S_1, S_2 \in \mathcal{M}$ such that U is a suffix of S_1 and S_2 and is preceded by different characters in S_1 and S_2.

The colexBWT

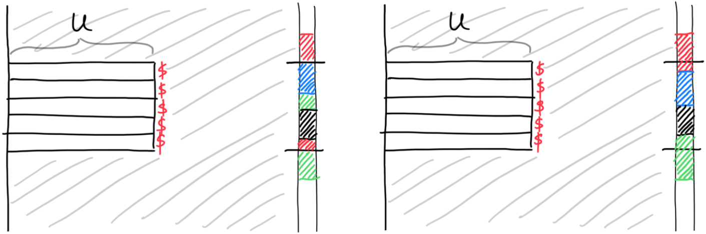
colexBWT: sort input strings colexicographically, then multidollarBWT



In the colexBWT, each interesting interval has at most σ runs.

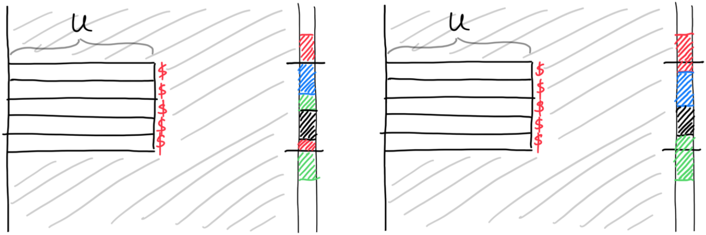
The optimalBWT

(Cenzato, Guerrini, L., Rosone, DCC 2023)

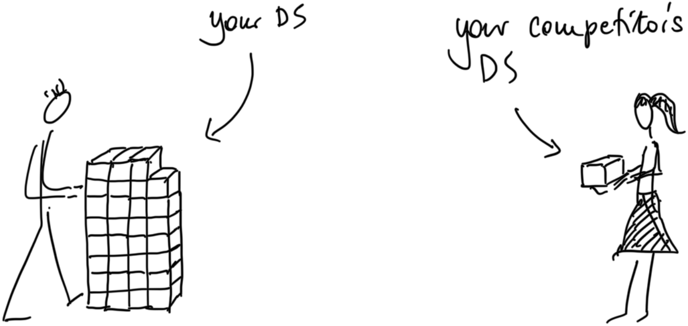


The optimalBWT

(Cenzato, Guerrini, L., Rosone, DCC 2023)



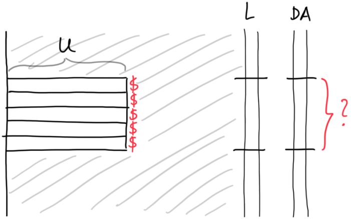
- complication due to successive interesting intervals
- based on algorithm by Bentley, Gibney, Thankachan (ESA, 2020)
- we implemented it, combining it with SAIS and BCR
- negligible computational overhead



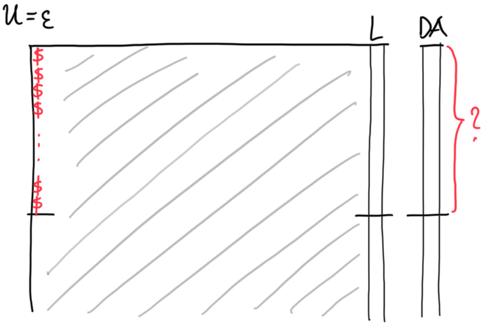
Improvement by optimalBWT on real biological data:

- in Cenzato & L. (2022, 2024): multipl. factor of up to 4.2
- in Guerrini, Cenzato, L., Rosone (2023): – "– of up to 31.5

What is the output order of the concatBWT?



Cenzato, L., Masillo, Rossi, forthcoming

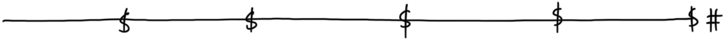


Observation

- Let $U = \epsilon$. Then the U-interval is $[1, k]$, where $k = |\mathcal{M}|$.
- k-prefix of the DA = output order.
- The order in all other interesting intervals is induced by this.

What is the output order of the concatBWT?

Concat BWT (same dollar plus # < $)



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

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

rotation	concatBWT	DA
\$#ATATG\$TGA\$ACG\$ATCA\$GGA	A	5
\$ACG\$ATCA\$GGA\$#ATATG\$TGA	A	2
\$ATCA\$GGA\$#ATATG\$TGA\$ACG	G	3
\$GGA\$#ATATG\$TGA\$ACG\$ATCA	A	4
\$TGA\$ACG\$ATCA\$GGA\$#ATATG	G	1
...	...	...

Map the strings to their lexicographic rank:

ACG	↦	a
ATATG	↦	b
ATCA	↦	c
GGA	↦	d
TGA	↦	e

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

input: b e a c d # **output:** d e a c b ($DA : 5, 2, 3, 4, 1$)

Map the strings to their lexicographic rank:

ACG	↦	a
ATATG	↦	b
ATCA	↦	c
GGA	↦	d
TGA	↦	e

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

input: b e a c d # **output:** d e a c b $(DA : 5, 2, 3, 4, 1)$

We realized that this is the BWT of the metacharacter-string! (almost)

Map the strings to their lexicographic rank:

ACG	↦	a
ATATG	↦	b
ATCA	↦	c
GGA	↦	d
TGA	↦	e

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

input: b e a c d # **output:** d e a c b ($DA: 5, 2, 3, 4, 1$)

We realized that this is the BWT of the metacharacter-string! (almost)

$$\begin{matrix} b & e & a & c & d & \# \\ e & a & c & d & \# & b \\ a & c & d & \# & b & e \\ c & d & \# & b & e & a \\ d & \# & b & e & a & c \\ \# & b & e & a & c & d \end{matrix} \quad \xrightarrow[\text{order}]{\text{lexicographic}} \quad \begin{matrix} \# & b & e & a & c & d \\ a & c & d & \# & b & e \\ b & e & a & c & d & \# \\ c & d & \# & b & e & a \\ d & \# & b & e & a & c \\ e & a & c & d & \# & b \end{matrix}$$

Map the strings to their lexicographic rank:

ACG	↦	a
ATATG	↦	b
ATCA	↦	c
GGA	↦	d
TGA	↦	e

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

input: b e a c d # **output:** d e a c b $(DA: 5, 2, 3, 4, 1)$

We realized that this is the BWT of the metacharacter-string! (almost)

$$\begin{array}{cccccc} \texttt{b} & \texttt{e} & \texttt{a} & \texttt{c} & \texttt{d} & \texttt{\#} \\ \texttt{e} & \texttt{a} & \texttt{c} & \texttt{d} & \texttt{\#} & \texttt{b} \\ \texttt{a} & \texttt{c} & \texttt{d} & \texttt{\#} & \texttt{b} & \texttt{e} \\ \texttt{c} & \texttt{d} & \texttt{\#} & \texttt{b} & \texttt{e} & \texttt{a} \\ \texttt{d} & \texttt{\#} & \texttt{b} & \texttt{e} & \texttt{a} & \texttt{c} \\ \texttt{\#} & \texttt{b} & \texttt{e} & \texttt{a} & \texttt{c} & \texttt{d} \end{array} \quad \xrightarrow[\text{order}]{\text{lexicographic}} \quad \begin{array}{cccccc} \texttt{\#} & \texttt{b} & \texttt{e} & \texttt{a} & \texttt{c} & \texttt{d} \\ \texttt{a} & \texttt{c} & \texttt{d} & \texttt{\#} & \texttt{b} & \texttt{e} \\ \texttt{b} & \texttt{e} & \texttt{a} & \texttt{c} & \texttt{d} & \texttt{\#} \\ \texttt{c} & \texttt{d} & \texttt{\#} & \texttt{b} & \texttt{e} & \texttt{a} \\ \texttt{d} & \texttt{\#} & \texttt{b} & \texttt{e} & \texttt{a} & \texttt{c} \\ \texttt{e} & \texttt{a} & \texttt{c} & \texttt{d} & \texttt{\#} & \texttt{b} \end{array}$$

output order: $\text{bwt}(\texttt{beacd\#}) = \texttt{de\#acb} \quad \rightsquigarrow \quad \texttt{deacb}$

- the output order of the concatBWT is the BWT of the meta-string of the input (almost)

- the output order of the concatBWT is the BWT of the meta-string of the input (almost)
- on most datasets, the concatBWT and the multidolBWT will differ

- the output order of the concatBWT is the BWT of the meta-string of the input (almost)
- on most datasets, the concatBWT and the multidolBWT will differ
- the concatBWT cannot produce all BWT variants:

k	3	4	5	6	7	8	9	10	11
	83.33%	75.0%	68.33%	63.89%	60.12%	57.29%	54.8%	52.81%	51.0%

- the output order of the concatBWT is the BWT of the meta-string of the input (almost)
- on most datasets, the concatBWT and the multidolBWT will differ
- the concatBWT cannot produce all BWT variants:

k	3	4	5	6	7	8	9	10	11
	83.33%	75.0%	68.33%	63.89%	60.12%	57.29%	54.8%	52.81%	51.0%

- only those which, inserting # somewhere, can become the BWT of some meta-string

- the output order of the concatBWT is the BWT of the meta-string of the input (almost)
- on most datasets, the concatBWT and the multidolBWT will differ
- the concatBWT cannot produce all BWT variants:

k	3	4	5	6	7	8	9	10	11
	83.33%	75.0%	68.33%	63.89%	60.12%	57.29%	54.8%	52.81%	51.0%

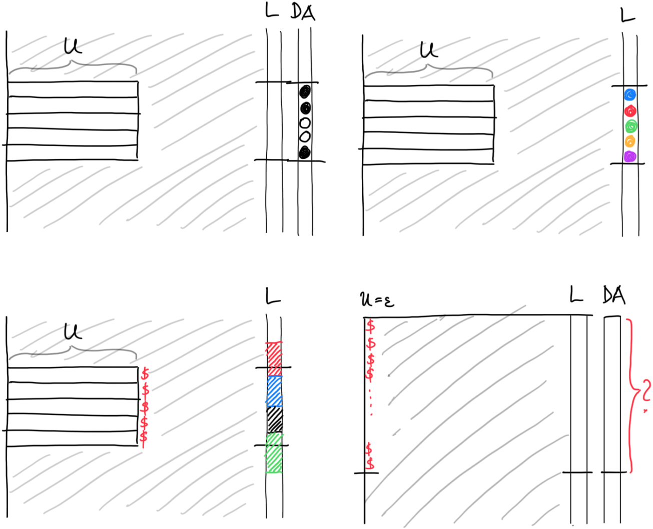
- only those which, inserting # somewhere, can become the BWT of some meta-string
- examples already on 3 strings where it cannot produce the optimalBWT

- the output order of the concatBWT is the BWT of the meta-string of the input (almost)
- on most datasets, the concatBWT and the multidolBWT will differ
- the concatBWT cannot produce all BWT variants:

k	3	4	5	6	7	8	9	10	11
	83.33%	75.0%	68.33%	63.89%	60.12%	57.29%	54.8%	52.81%	51.0%

- only those which, inserting # somewhere, can become the BWT of some meta-string
- examples already on 3 strings where it cannot produce the optimalBWT
- a first study of strings which are the bwt* of some string in (Giuliani, L., Masillo, Rizzi: *When a dollar makes a BWT*, TCS 2021)

Summary (BWT everywhere)



Conclusions

1. There is more to the BWT than just compression.

Conclusions

1. There is more to the BWT than just compression.
   - For instance, it can be used to generate random de Bruijn sequences.

Conclusions

1. There is more to the BWT than just compression.
   - For instance, it can be used to generate random de Bruijn sequences.
2. It makes a difference how the BWT of a string collection is computed.

Conclusions

1. There is more to the BWT than just compression.
   - For instance, it can be used to generate random de Bruijn sequences.
2. It makes a difference how the BWT of a string collection is computed.
   - do not use the concatBWT.
   - use the multidollarBWT or the original eBWT.
   - even better: use the optimalBWT.

Conclusions

1. There is more to the BWT than just compression.
   - For instance, it can be used to generate random de Bruijn sequences.
2. It makes a difference how the BWT of a string collection is computed.
   - do not use the concatBWT.
   - use the multidollarBWT or the original eBWT.
   - even better: use the optimalBWT.
3. Definition of the number of runs r for string collections should be standardized (optBWT or colexBWT).

Acknowledgements

Massimiliano Rossi

Sara Giuliani

Davide Cenzato

Francesco Masillo

Luca Parmigiani

Veronica Guerrini

Giovanna Rosone

rukrn!h t Ttnoaeifyyuotnaoo

rukrn!h t Ttnoaeifyyuotnaoo

s?utoinesQ