XBWTing Readsets

Taking advantage of read alignment to obtain better compression

Travis Gagie, **Garance Gourdel**, Giovanni Manzini

DALHOUSIE UNIVERSITY
IRISA
Inria
ENS ÉCOLE NORMALE SUPÉRIEURE
GENSCALE
UPO UNIVERSITÀ DEL PIEMONTE ORIENTALE

The FM-Index *On Strings*

Compresses and **indexes** the input.

Applied in short read aligners such as **Bowtie** and **BWA**.

Recently improved to support in O(r) space, with **r the number of runs in the BWT** :

- **count(P)**: counting the occ. of a pattern P in **O(|P|)** time.
- **locate(P)**: locate the occ. Of P in **O(|P|+occ)** time.

[Gagie, Navarro, and Prezza., Fully functional suffix trees and optimal text searching in bwt-runs bounded space]

Example:

`S = GATTAGATACAT$`

`BWT(S) = TTTCGGAA$AATA` => 8 runs.

	F … L
0	$GATTAGATACAT
1	ACAT$GATTAGAT
2	AGATACAT$GATT
3	AT$GATTAGATAC
4	ATACAT$GATTAG
5	ATTAGATACAT$G
6	CAT$GATTAGATA
7	GATACAT$GATTA
8	GATTAGATACAT$
9	T$GATTAGATACA
10	TACAT$GATTAGA
11	TAGATACAT$GAT
12	TTAGATACAT$GA

Can we have the **same number of runs** for reads extracted from S ?

The FM-Index for Readsets *Naive approach*

Concatenate the reads with a separator “$” and build the FM-index of the entire string.

Example:

```
S = GATTA$TTAGA$TAGATA$GATAC$ATACAT$
BWT(S) = CATAATGTTTTTCGG$GAAAA$$AATAAT$A$
```
=> 20 runs.

Issues:

- Computing the BWT for such a long string is challenging (and the context before the dollars is not relevant).
- The $ break some runs as in `CGG$G` in the example.

The FM-Index for Readsets *EBWT*

Extended BWT: Permute the characters in the strings into the lexicographic order of the suffixes that immediately follow them, considering each string to be cyclic.

Example:

```
S = {GATTA$, TTAGA$, TAGATA$, GATAC$, ATACAT$}
EBWT(S) = TCAAATTGTTTTCGG$GAAAA$$ATAAAT$A$
```
=> 19 runs.

Easier to build and update than the naive approach.

Issues:

- Still has more than double the number of run than `BWT(GATTAGATACAT)`.

Heuristics for read ordering for the EBWT

RLO: Reorganizing the reads in reversed lexicographic order (co-lexicographic order)

Example:

```
    RLO(S) = {GATTA$, TTAGA$, TAGATA$, GATAC$, ATACAT$}
    EBWT(RLO(S)) = AAACTGTTTTTTCGG$GAAAA$$AATAAT$A$ => 19 runs
```

SPRING: attempts to reorder reads according to their position in the genome.

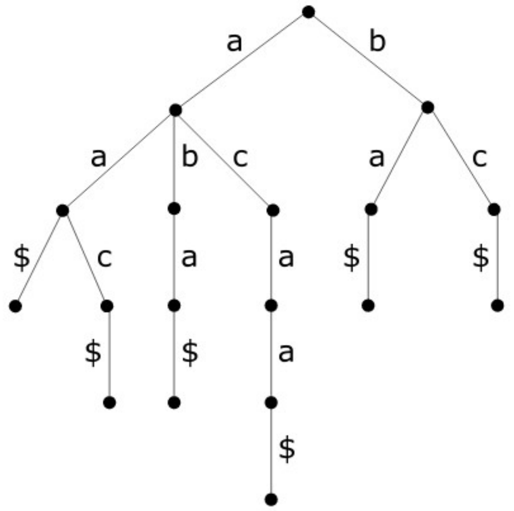
```
    EBWT(SPRING(S))= ACATATTGTTTTCGG$GAAAA$$ATAAAT$A$  => 22 runs
```

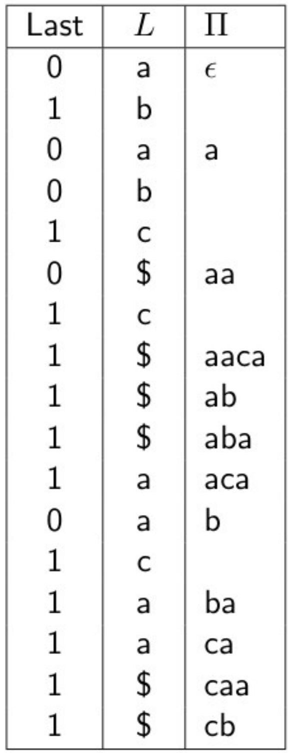
The FM-Index for Readsets

XBWT

eXtended BWT: Generalization of the BWT for labeled trees where the characters get sorted according to the label of there outgoing node.

Can also be seen as a sub-case of a Wheeler graph.





Last	L	Π
0	a	ϵ
1	b	
0	a	a
0	b	
1	c	
0	$	aa
1	c	
1	$	aaca
1	$	ab
1	$	aba
1	a	aca
0	a	b
1	c	
1	a	ba
1	a	ca
1	$	caa
1	$	cb

Figure: XBWT Tricks, *Giovanni Manzini*

Our approach

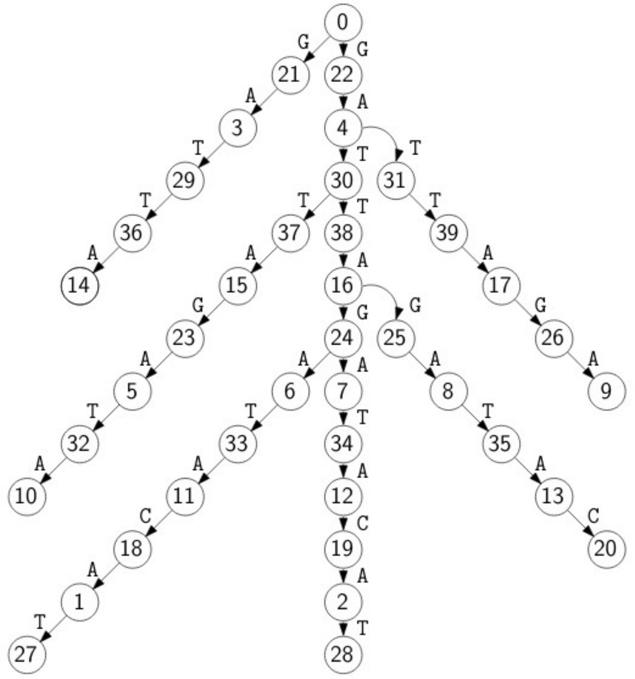
Example

If the reads correspond to a known assembled genome, use the genome as additional context for better compression.

Example:

```
  TTAGA
   TAGATA
GATTAGATACAT
GATTA ATACAT
     GATAC
```

T =



XBWT(T)=GGTTTTTTTTTCCCGGGGAAAAAAAAATTTTAAAAAAAA => 7 runs.

Our approach

Theorem 1

If we create such a tree, where the **reads are errorlessly sampled and aligned to the reference**, then the XBWT of the tree has the **same number of runs** as the BWT of the reverse of the reference.

Our approach

Theorem 2

If we create a labeled tree T as explained, let:
r be the number of runs in the XBWT,
t be the number of reads,
Then in O(r+t) words of space,
We can compute Locate(P) in O((|P|+occ)log logn)

In reality the reads we have to index are **not perfectly matching** the assembled genome.
How does this approach to compression work in practice ?

Our approach

PFP construction of the XBWT

Scan

- Scan: Slide a window, compute the KR-fingerprint of the window, cutting the text into prefix free phrases
- Scan the reference creating a parse and a dictionnary
- For each read, extend the read, parse the extended read

XBWT of the parse

- Doubling algorithm to determine the XBWT of the parse

Final XBWT

- SA and LCP of all suffixes of the words in the dictionary
- For each suffix, add the correct character depending on the cases

Only on the number of runs for now.

We compared to:

- **EBWT** (using the ropebwt2 implemetation), with and without \$.
- **SPRING + EBWT,** with and without \$.
- **RLO + EBWT,** with and without \$.

EBWT > SPRING + EBWT > RLO +EBWT

So in the graphs we focus on RLO+EBWT.

Experimental results

On the following **datasets**:

- **synthetic**: generated reads with controllable error rates
- **Real world:** only reads matched to the genome, not the reverse complement
    - **E.coli**: from the single cell dataset
    - **Stapphylococcus aureus**: from the single cell dataset
    - **Hydrophilia**: from the Gage-b dataset

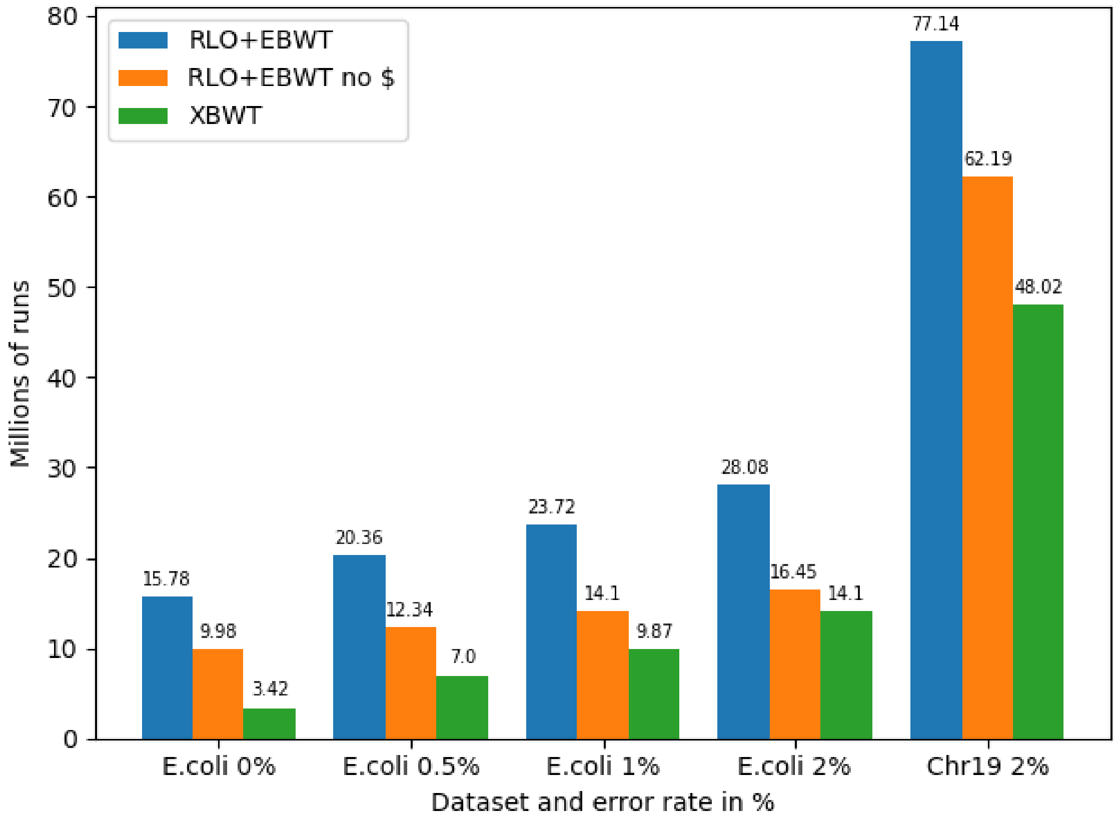
Our approach

Experimental results

Synthetic data

Generated by wgsim:

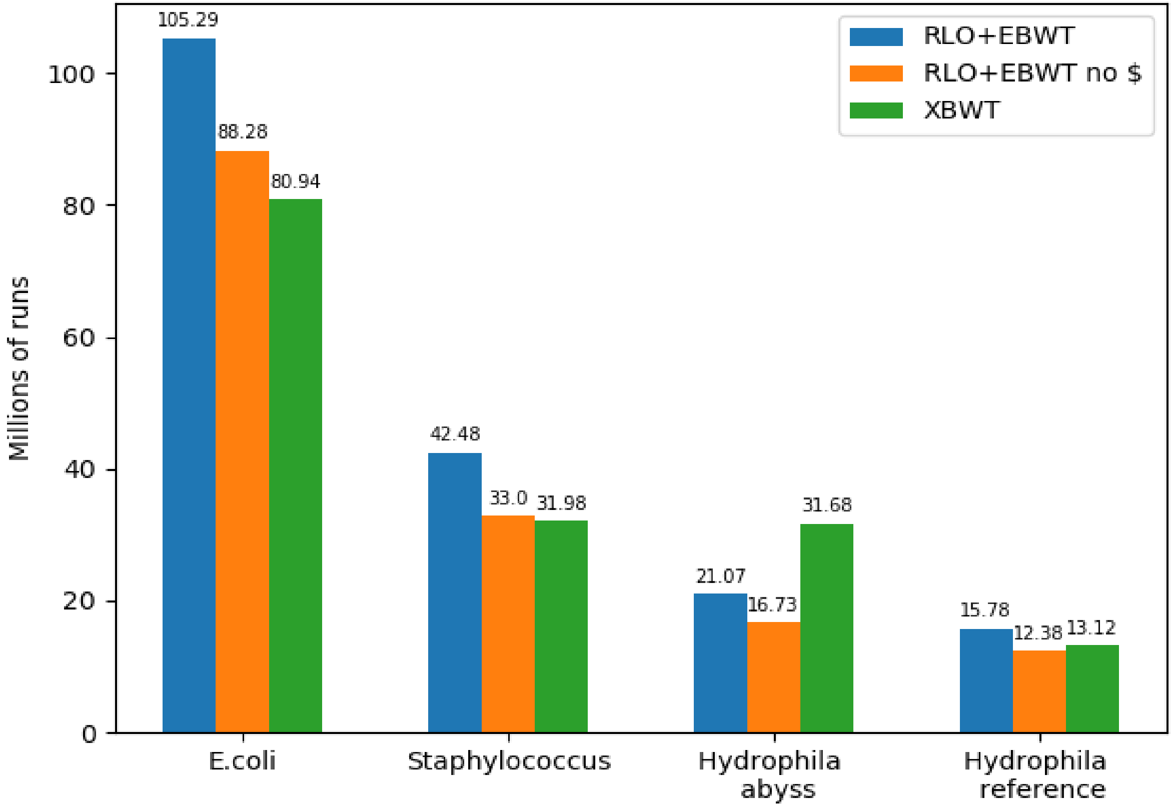
- Error rate from 0 to 2%
- 20 Millions reads of length 100 bp
- 15% of polymorphisms are INDELs with their lengths drawn from a geometric distribution
- The XBWT is always more compact, even for 2% >> 0.1%.



Our approach | *Experimental results*

Real data

- Good or similar for E.coli and Staphylococcus.
- Surprising result for hydrophila?
- For E. coli readset, RLO+EBWT without $ has 16% less runs than with $ and the XBWT 8.3% less run than RLO+EBWT no $.



To do:

- Evaluation on the human Genome
- FM-index, implementation and time analysis
- Time comparison of PFP construction of the XBWT compared to other construction [BWT-tunneling by Uwe Baier, Wheeler sort by Jarno Alanko]

Future work

Hybrid index improvement?

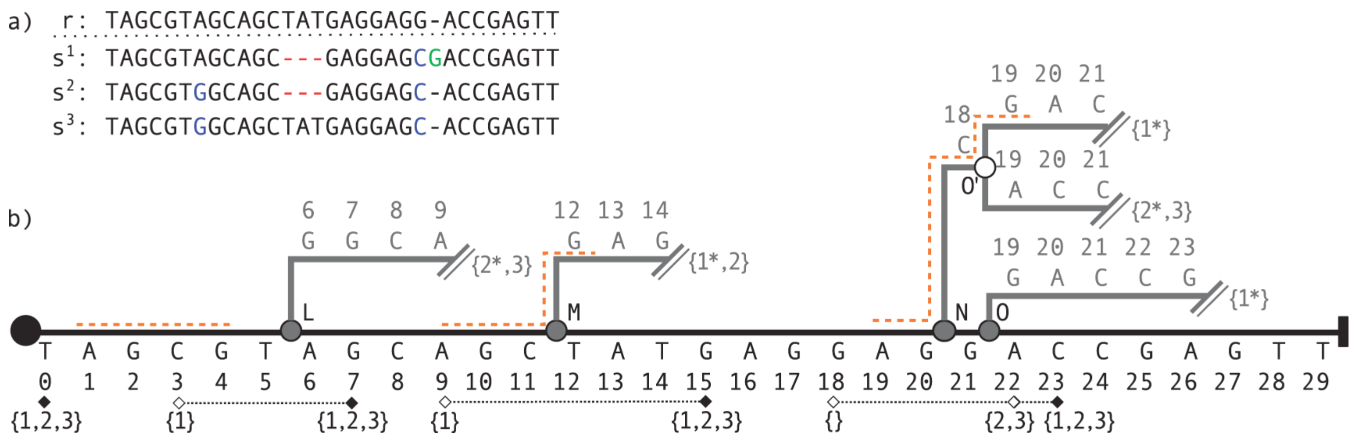

a)
r: TAGCGTAGCAGCTATGAGGAGG-ACCGAGTT
s1: TAGCGTAGCAGC---GAGGAGCGACCGAGTT
s2: TAGCGTGGCAGC---GAGGAGC-ACCGAGTT
s3: TAGCGTGGCAGCTATGAGGAGC-ACCGAGTT
b)
L
M
N
O
O'
{2*,3}
{1*,2}
{1*}
{2*,3}
{1*}
{1,2,3}
{1}
{1,2,3}
{1}
{1,2,3}
{}
{2,3}
{1,2,3}

Figure: *René Rahn, David Weese, and Knut Reinert*
Journaled string tree

Storing the XBWT of a JST could be more efficient than storing the BWT of a string kernel.

Thank you for your attention !

Take away message:

- For compression removing the dollars is important.
- Taking advantage of the genome for additional context brings a better compression!

Questions ?

Pre-publication: https://arxiv.org/pdf/1809.07320.pdf