



UNIVERSITÀ
di VERONA



Università
degli Studi
di Palermo

UF | UNIVERSITY of
FLORIDA

Computing the original eBWT faster, simpler, and with less memory

Christina Boucher¹, Davide Cenzato², Zsuzsanna Lipták², Massimiliano Rossi¹, Marinella Sciortino³

¹ University of Florida, Department of Computer & Information Science & Engineering.

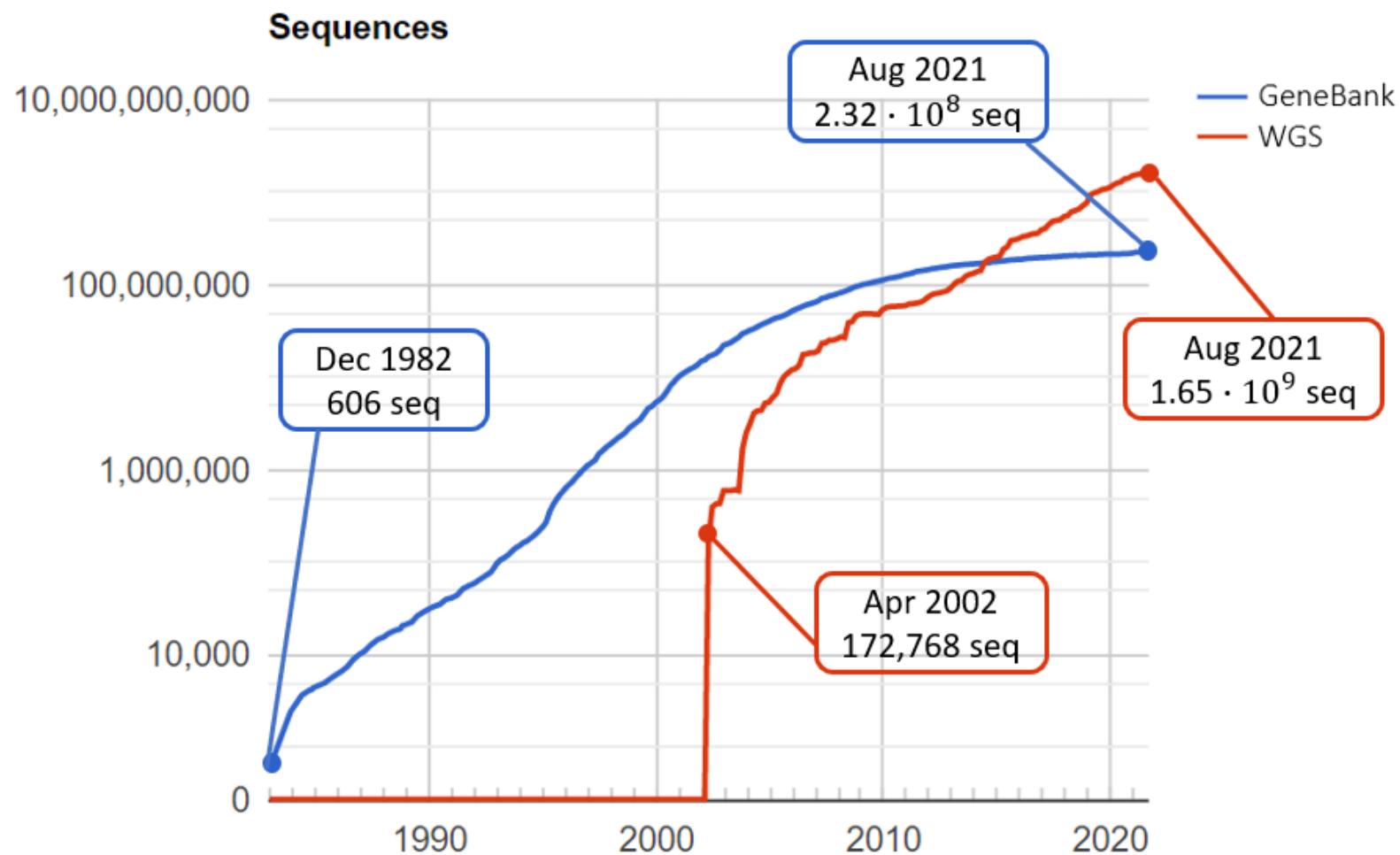
² University of Verona, Department of Computer Science.

³ University of Palermo, Department of Mathematics and Computer Science.

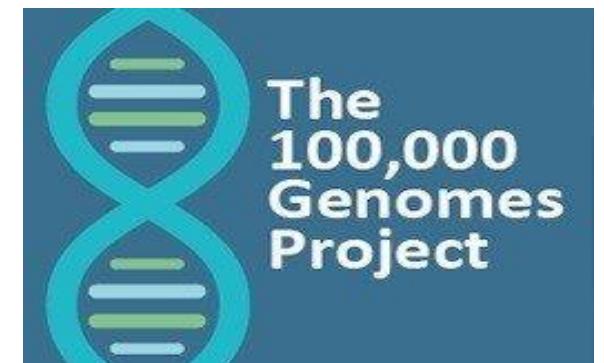
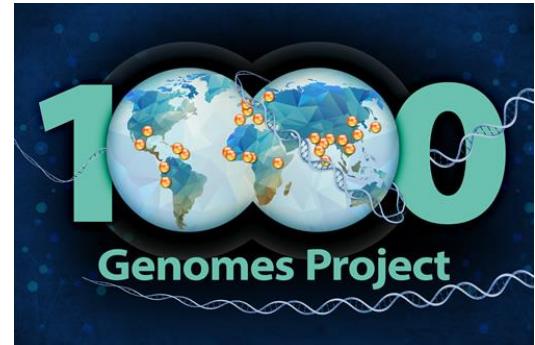


SPIRE 2021, October 5th, 2021 - Lille, France (Online)

Motivation



source: <https://www.ncbi.nlm.nih.gov/genbank/statistics/>



The Burrows-Wheeler Transform (BWT) is a highly useful tool

- allows to compress the input while supporting efficient querying
- part of key tools in bioinformatics such as Bowtie and BWA

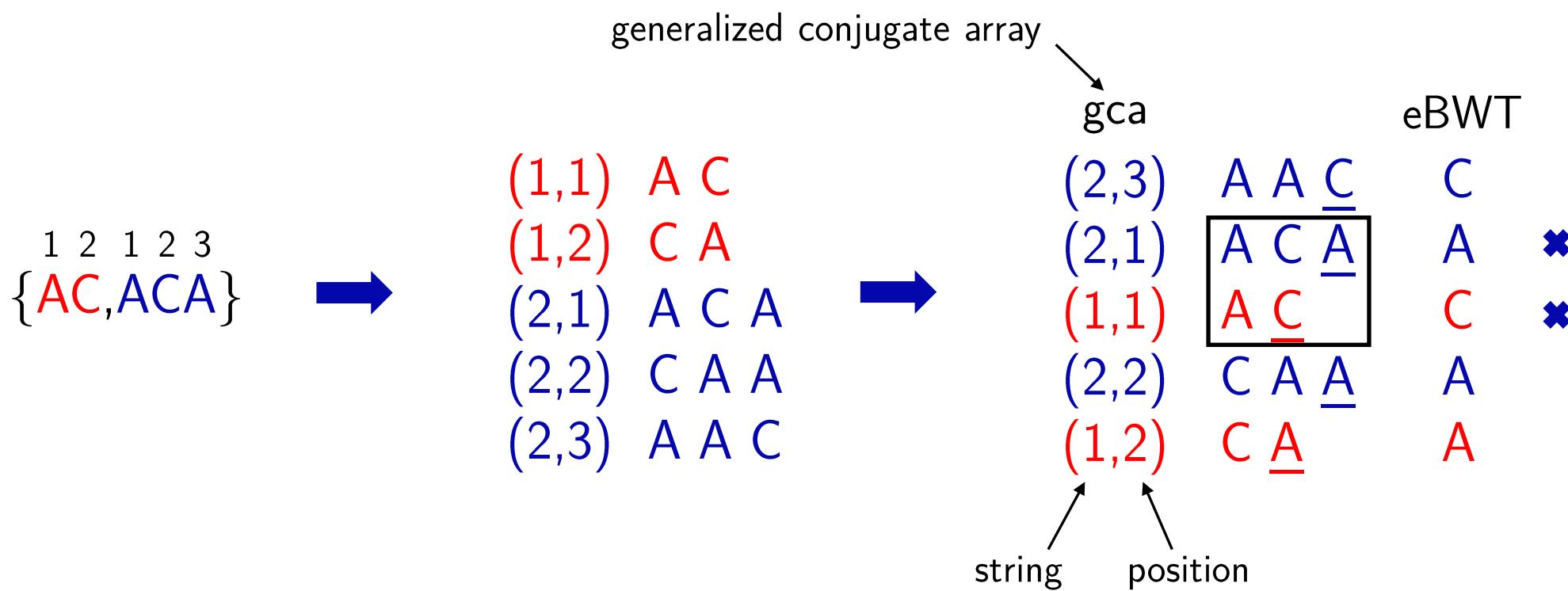
The extended BWT (eBWT)

- original BWT extension to string collections
- independent of the input order

- 1 simple linear time algorithm to compute the original eBWT
- 2 first linear time algorithm to compute the BWT of a single sequence using neither dollar nor Lyndon rotation
- 3 a combination of the new eBWT algorithm with a variant of Prefix-free parsing (PFP) to compute the eBWT of very large string collections

The extended BWT

The extended BWT (eBWT) of Mantaci et al. (2007) is a reversible transformation that takes in input a string collection \mathcal{M} producing a permutation of the characters in the strings of \mathcal{M} .



The omega-order ($<_\omega$)

Given two strings U and V , U is smaller than V according to the omega-order, $U <_\omega V$, if $U^\omega <_{lex} V^\omega$, where $U^\omega = UUU\dots$ and $V^\omega = VVV\dots$.

A C A A C A A ...
A C A C A C A ...

This definition can be generalized to non-primitive strings.

Our contributions

- 1 simple linear time algorithm to compute the original eBWT
- 2 first linear time algorithm to compute the BWT of a single sequence using neither dollar nor Lyndon rotation
- 3 a combination of the new eBWT algorithm with a variant of Prefix-free parsing (PFP) to compute the eBWT of very large string collections

- adaptation of the SAIS algorithm of Nong et al. (2011): **SAIS-for-eBWT**; important differences are:
 - sorting conjugates instead of suffixes
 - string collections instead of single strings
 - omega order instead of lexicographical order
 - no end-of-string characters
- similar to the algorithm of Bannai et al. (CPM 2021) for computing the eBWT (significantly simpler)

Computing cyclic types in linear time

Let T be a circular string of length at least 2, and $1 \leq i \leq |T|$

- i is **S type** if $\text{conj}_i(T) <_{\text{lex}} \text{conj}_{i+1}(T)$
- i is **L type** if $\text{conj}_i(T) >_{\text{lex}} \text{conj}_{i+1}(T)$
- an **S type** position i is called **LMS-position** (“leftmost S”, S^*) if $i - 1$ is **L type**

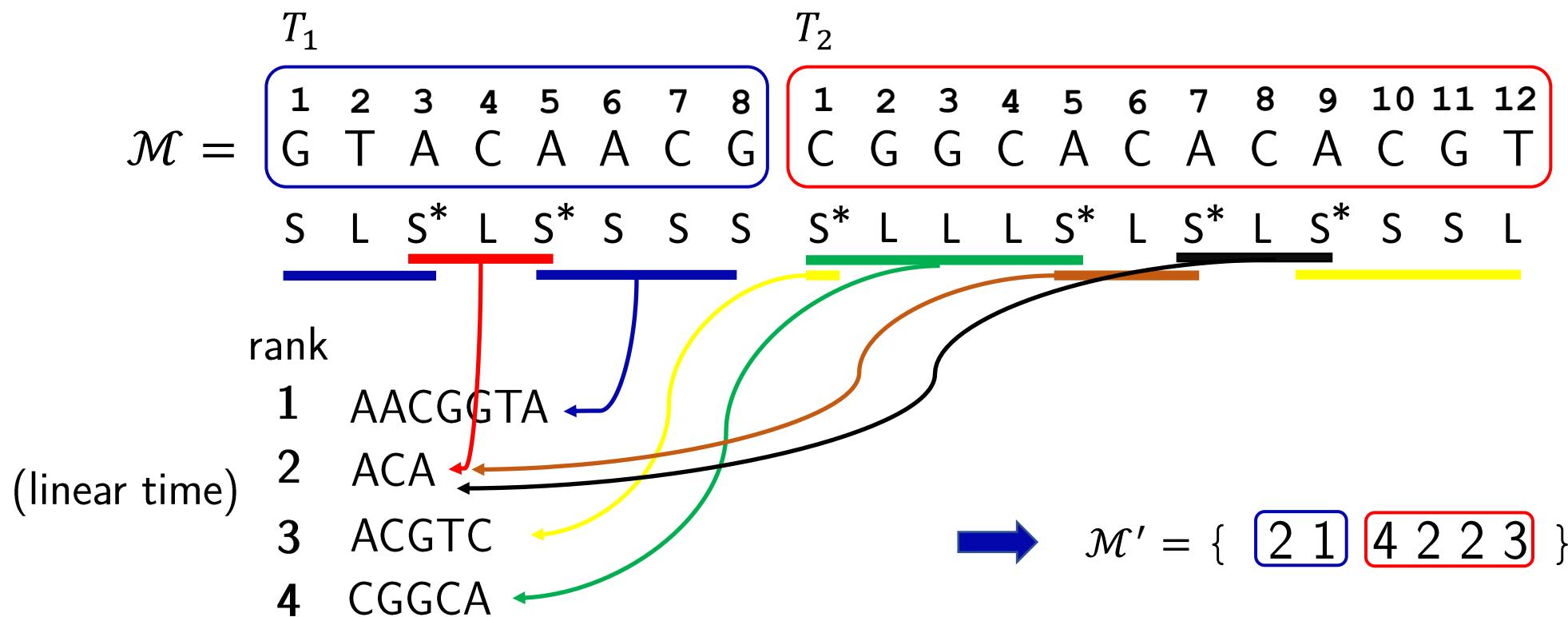
| | T_1 | | | | | | | | T_2 | | | | | | | |
|-----------------|-------|---|-------|---|-------|---|---|---|-------|---|---|---|-------|---|-------|---|
| $\mathcal{M} =$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| | G | T | A | C | A | A | C | G | C | G | G | C | A | C | A | C |
| | S | L | S^* | L | S^* | S | S | S | S^* | L | L | L | S^* | L | S^* | S |

Observation

Two scans for each string are sufficient to compute all types.

Sort and name the LMS-substrings

- sort the LMS-substrings
- build a new string collection using LMS-substrings' ranks



Induced sorting

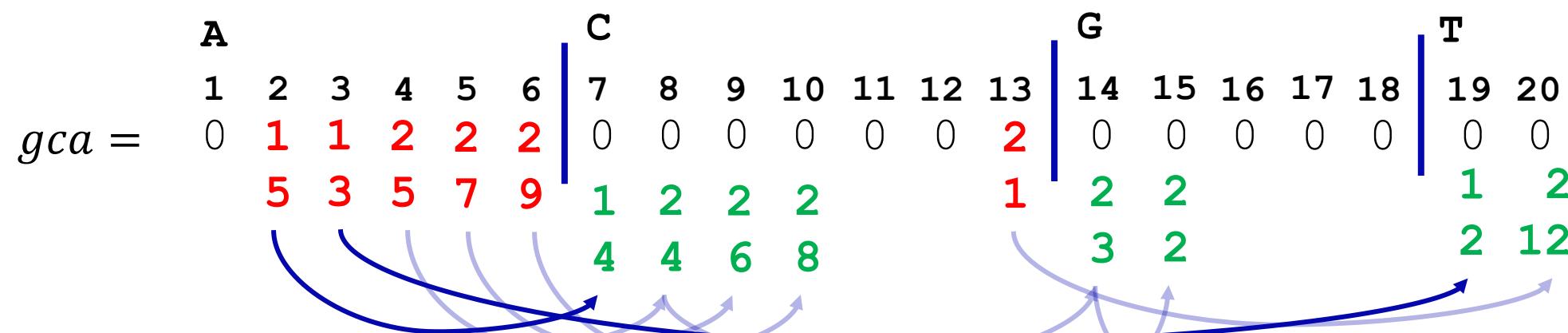
- map the correct positions of S^* from $gca(\mathcal{M}')$

| | T_1 | | | | | | | | T_2 | | | | | | | | | | | |
|-----------------|-------|---|-------|---|-------|---|---|---|-------|---|---|---|-------|---|-------|---|-------|----|----|----|
| \mathcal{M} = | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| | G | T | A | C | A | A | C | G | C | G | G | C | A | C | A | C | A | C | G | T |
| | S | L | S^* | L | S^* | S | S | S | S^* | L | L | L | S^* | L | S^* | L | S^* | S | S | L |

| gca = | A | C | G | T |
|---------|-------------|-------------------|----------------|-------------|
| | 1 2 3 4 5 6 | 7 8 9 10 11 12 13 | 14 15 16 17 18 | 19 20 |
| | 0 0 0 0 0 0 | 0 0 0 0 0 0 | 0 0 0 0 0 0 | 0 0 0 0 0 0 |
| | 1 1 2 2 2 | 2 | 1 | 0 0 0 0 0 0 |
| | 5 3 5 7 9 | | | 0 0 0 0 0 0 |

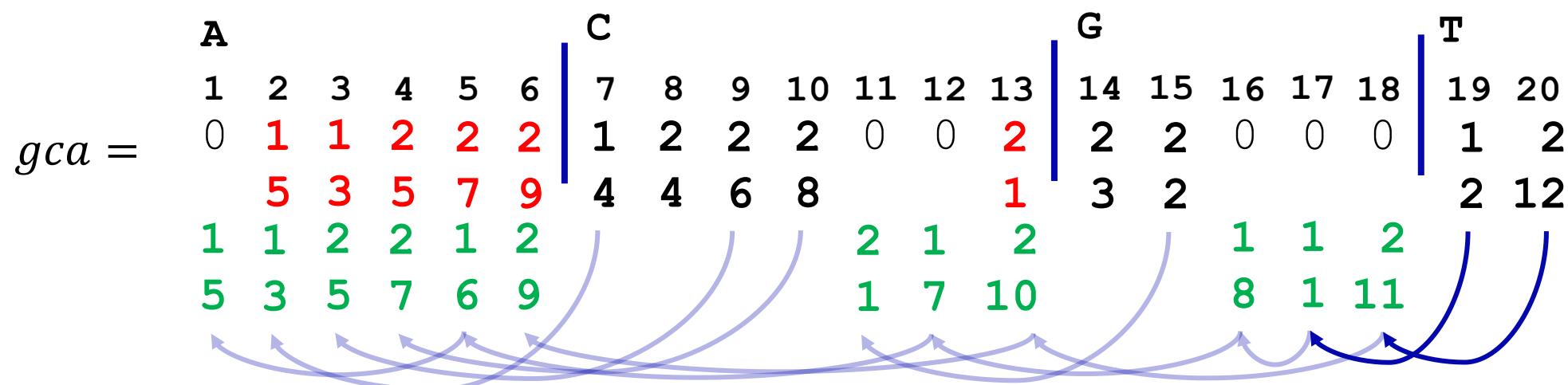
Induce L types

- induce L positions scanning gca from left to right

$$\mathcal{M} = \begin{array}{ccccccccc} T_1 & & T_2 & & & & & & \\ \boxed{\begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ G & T & A & C & A & A & C & G \end{matrix}} & & \boxed{\begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ C & G & G & C & A & C & A & C & A & C & G & T \end{matrix}} & & & & & & \\ S & L & S^* & L & S^* & S & S & S & S^* & L & L & L & S^* & L & S^* & L & S^* & S & S & S & L & L \end{array}$$


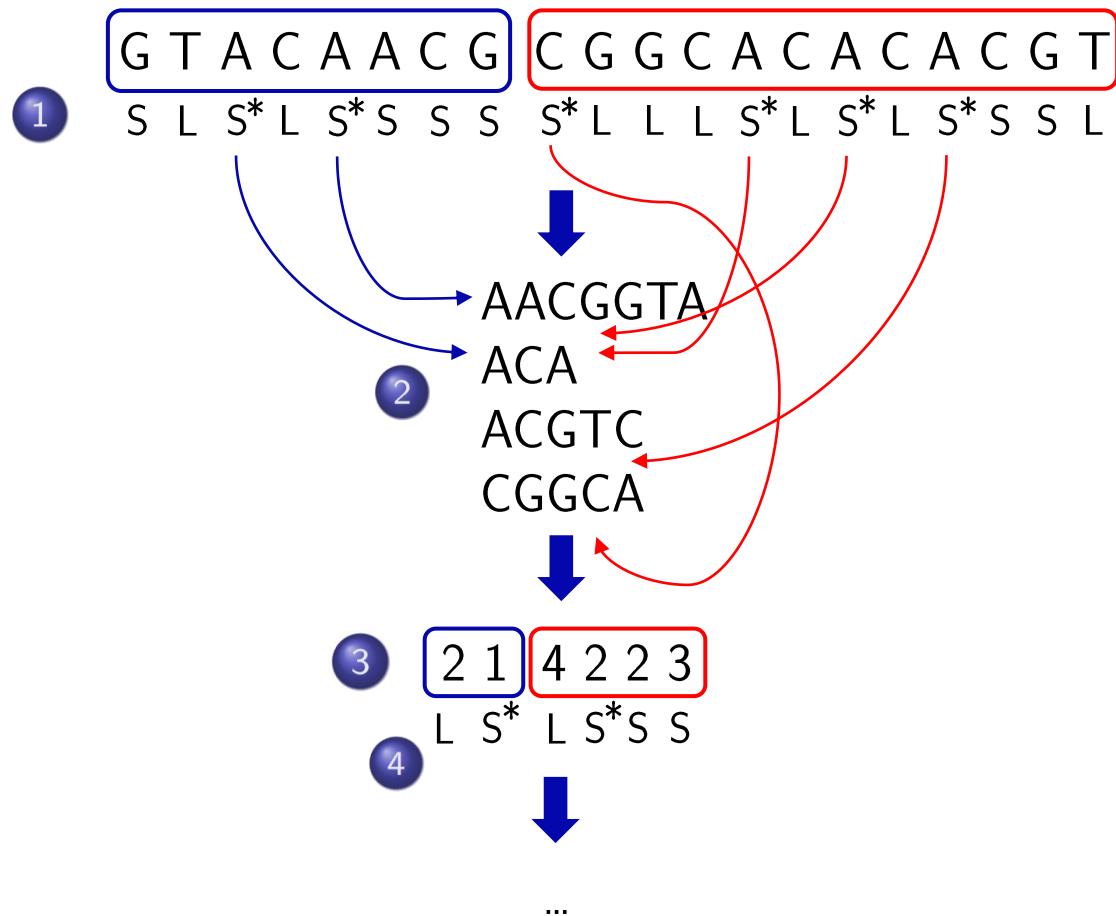
Induce S types

- induce S positions scanning gca from right to left

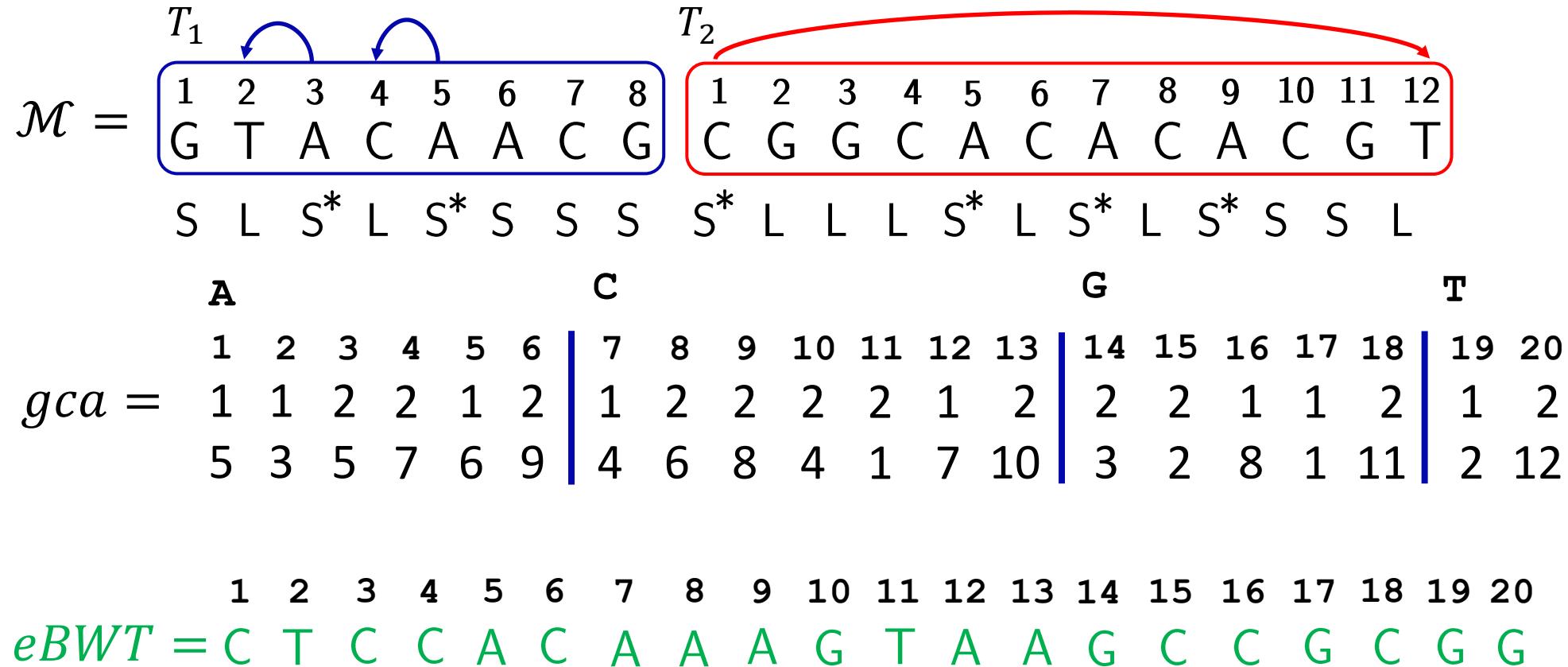
$$\mathcal{M} = \begin{array}{ccccccccc} T_1 & & T_2 & & & & & & \\ \boxed{\begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ G & T & A & C & A & A & C & G \end{matrix}} & & \boxed{\begin{matrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 \\ C & G & G & C & A & C & A & C & A & C & G & T \end{matrix}} & & & & & & \\ S & L & S^* & L & S^* & S & S & S & S^* & L & L & L & S^* & L & S^* & L & S^* & S & S & S & L \end{array}$$


High level view of the algorithm

- 1 compute cyclic types
- 2 sort the LMS-substrings
- 3 name the LMS-substrings
- 4 recurse
- 5 use induced sorting to sort all positions



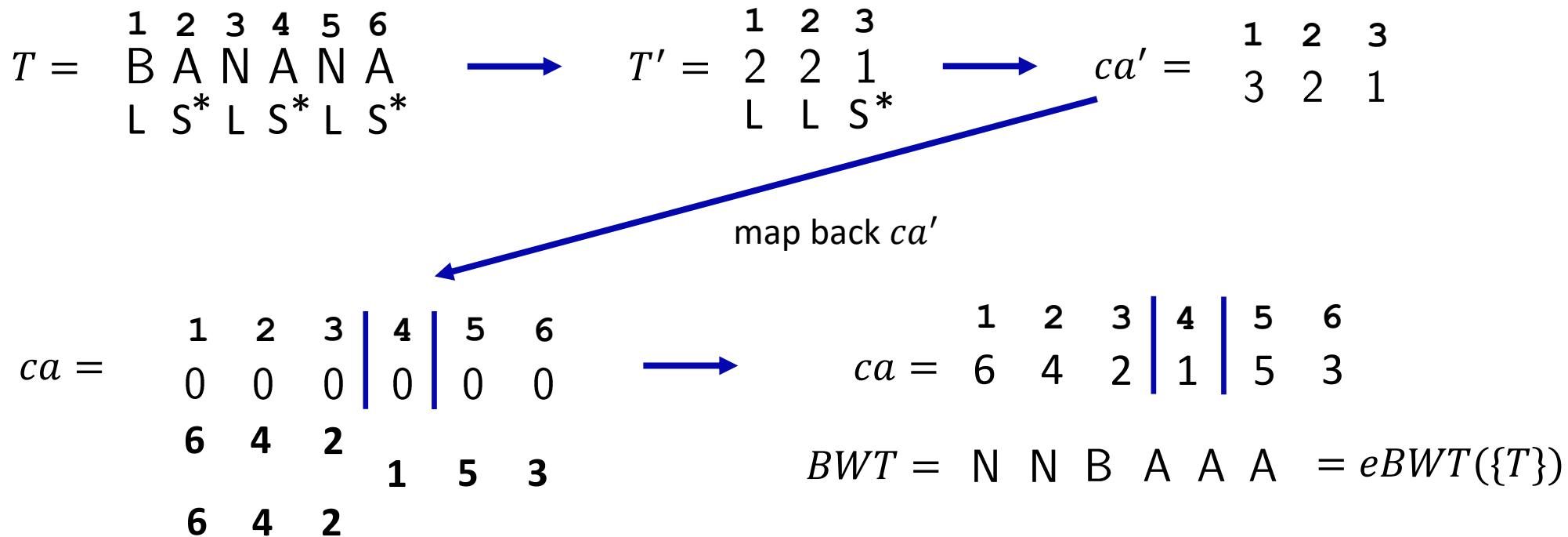
Compute the eBWT



Our contributions

- 1 simple linear time algorithm to compute the original eBWT
- 2 first linear time algorithm to compute the BWT of a single sequence using neither dollar nor Lyndon rotation
- 3 a combination of the new eBWT algorithm with a variant of Prefix-free parsing (PFP) to compute the eBWT of very large string collections

Building the BWT without \$



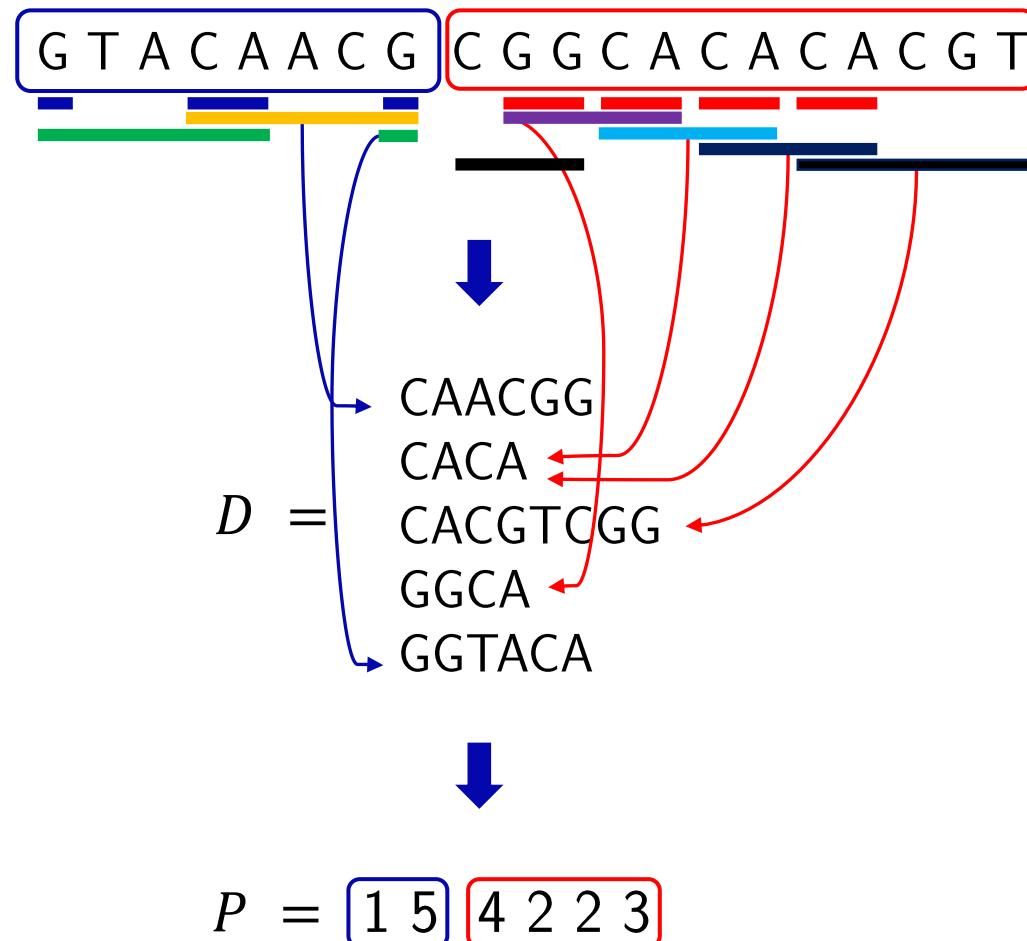
Our contributions

- 1 simple linear time algorithm to compute the original eBWT
- 2 first linear time algorithm to compute the BWT of a single sequence using neither dollar nor Lyndon rotation
- 3 a combination of the new eBWT algorithm with a variant of Prefix-free parsing (PFP) to compute the eBWT of very large string collections

- PFP is a preprocessing step introduced by Boucher et al. (WABI 2018) to ease the computation of the BWT of large and highly repetitive texts
 - computes a dictionary and a parse
 - BWT construction in space proportional to the size of the dictionary and parse
- We adapted the PFP to obtain the parse of string collections:
 - no concatenation and no end-of-string characters
 - dictionary phrases are defined circularly

Cyclic Prefix-free parsing (PFP)

- 1 compute the trigger strings
- 2 sort dictionary phrases
- 3 build the parse

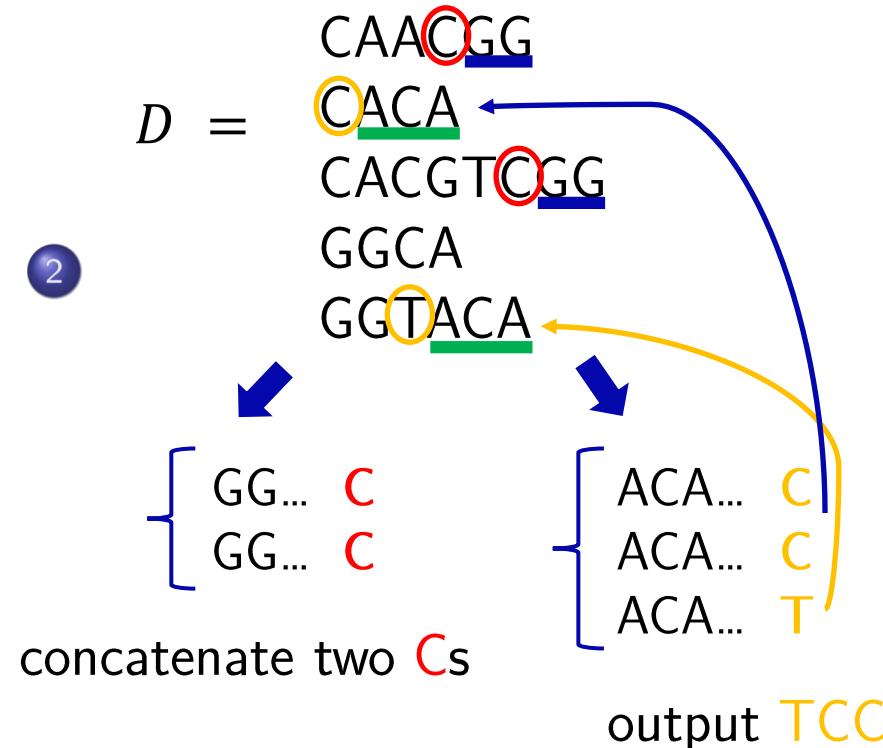


Using PFP to build the eBWT

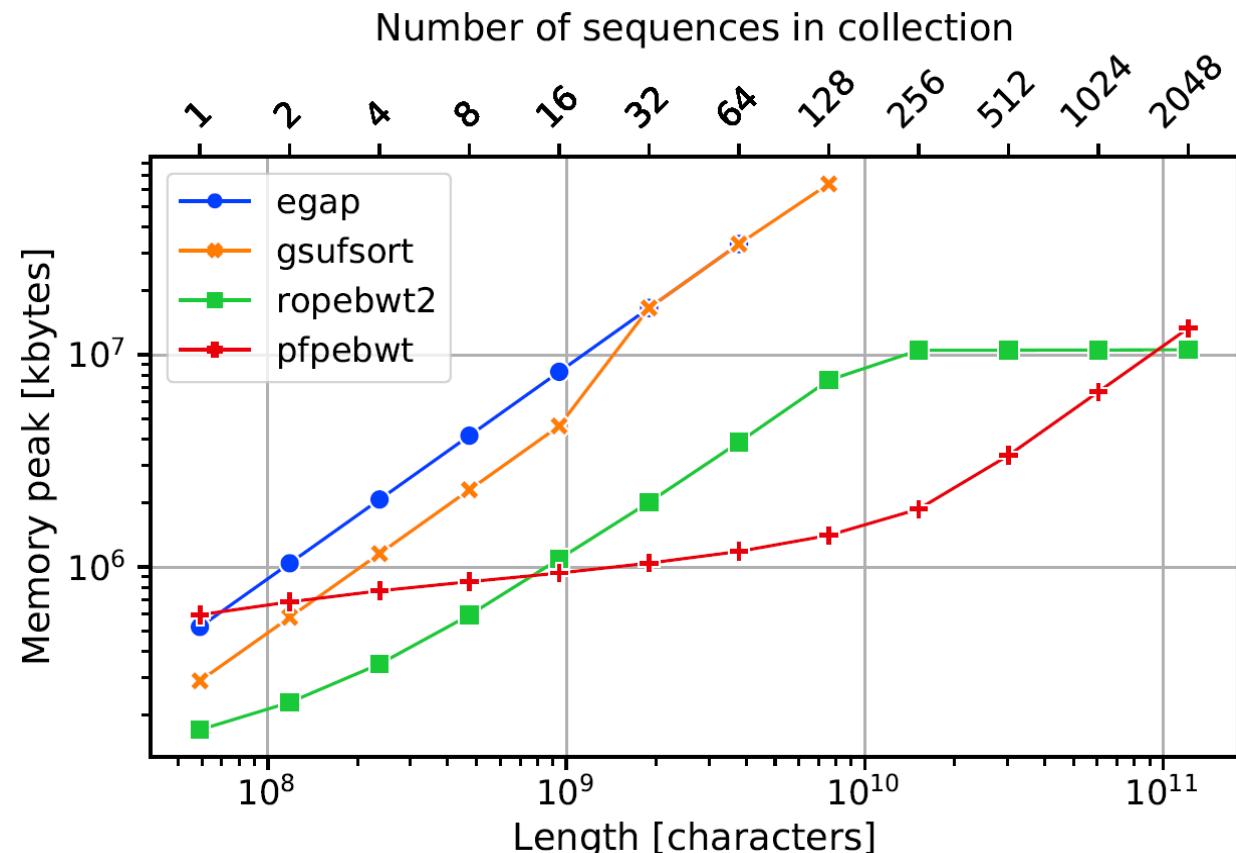
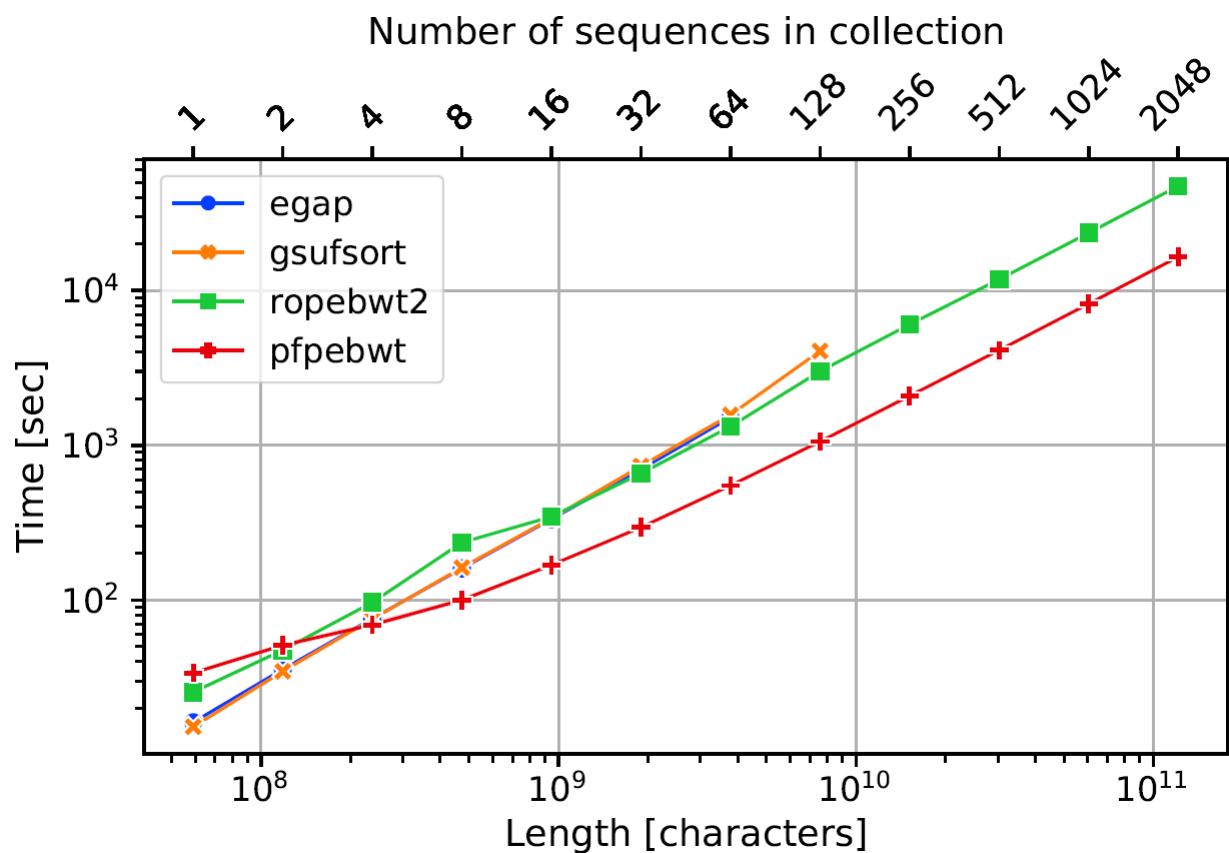
① compute the eBWT of P

② build the eBWT of the input collection

$$\text{① } \text{eBWT}(\{1\ 5, 4\ 2\ 2\ 3\}) = 5\ 4\ 2\ 2\ 3\ 1$$



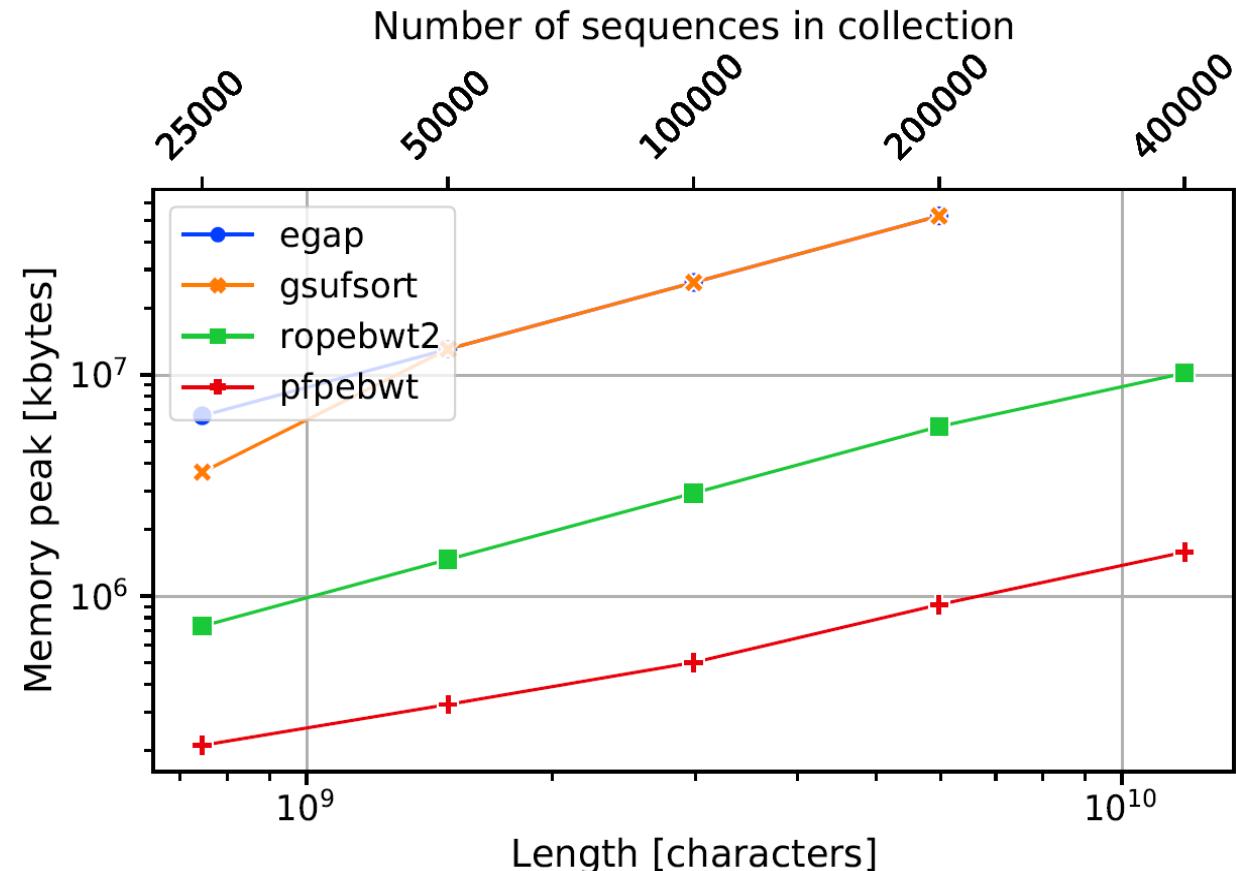
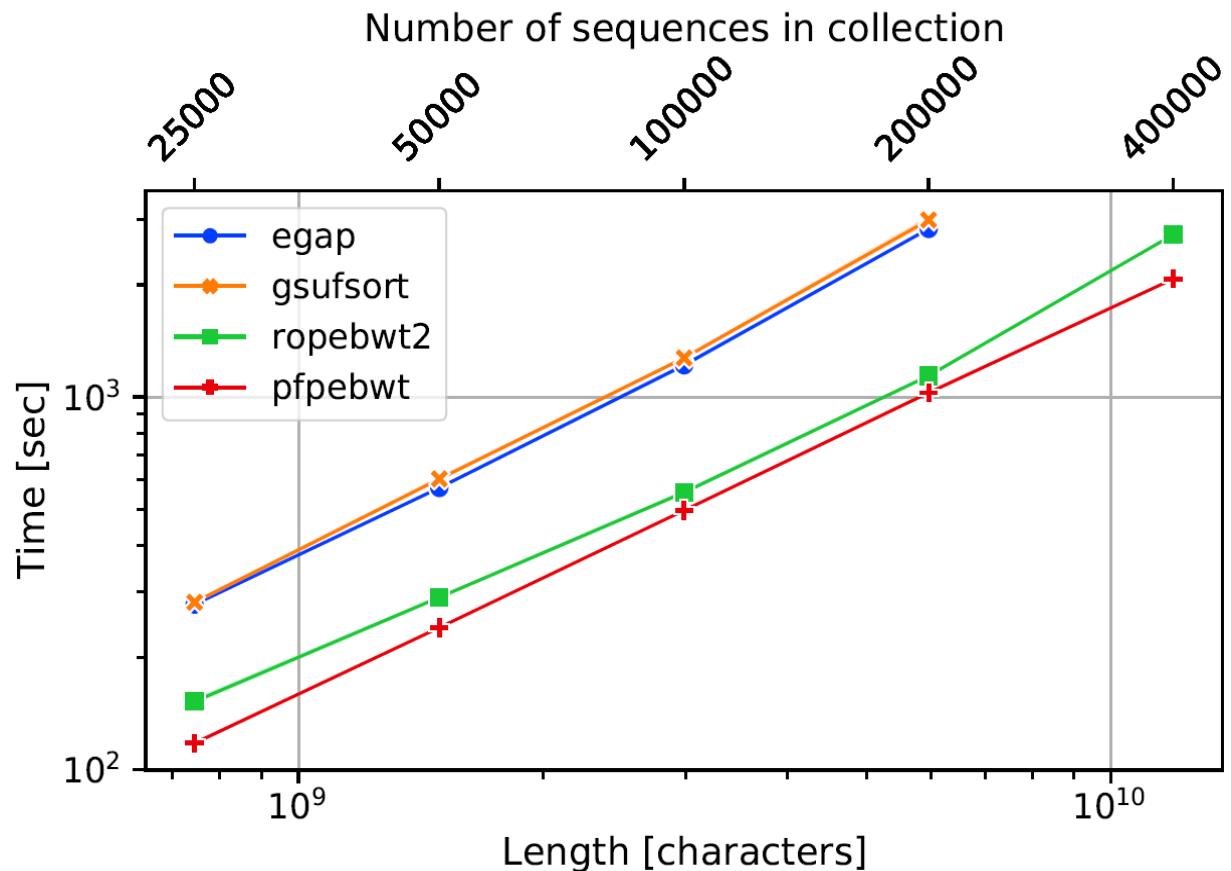
Experimental results – chr19



data from 1,000 Genomes Project

60 GB internal memory and 24 hours cut-off

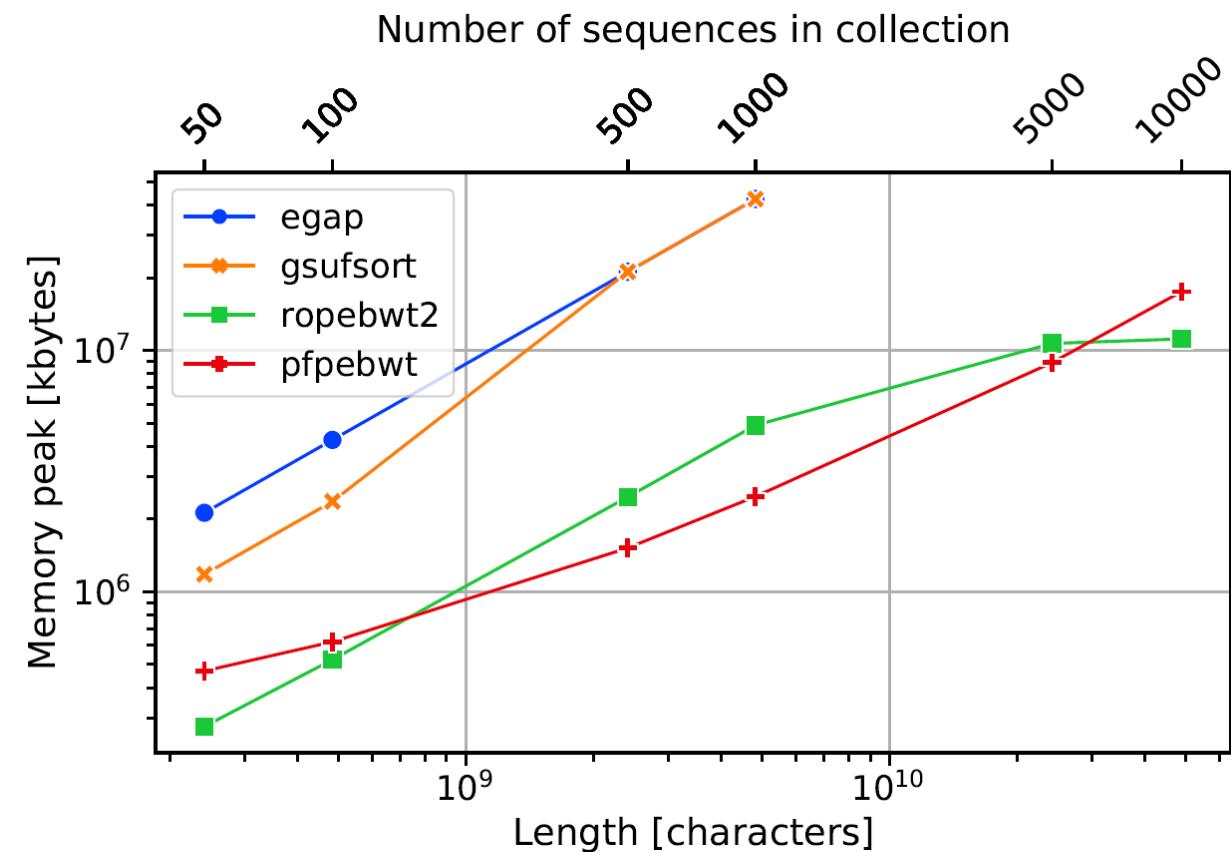
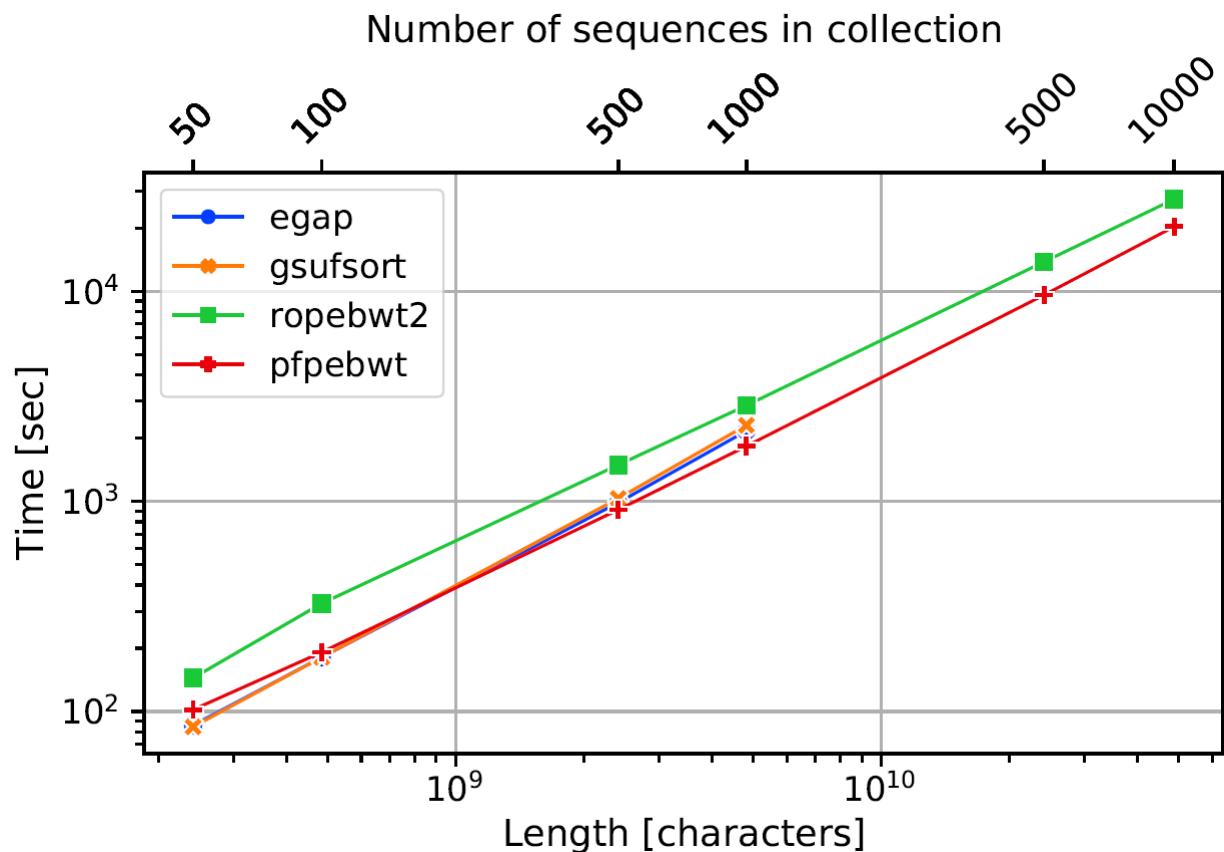
Experimental results – SARS-CoV-2



data from EBI's COVID-19 portal

60 GB internal memory and 24 hours cut-off

Experimental results – salmonella



data from the GenomeTrakr project

60 GB internal memory and 24 hours cut-off



UNIVERSITÀ
di VERONA



Università
degli Studi
di Palermo

UF | UNIVERSITY of
FLORIDA



additional funding by:

- NSF IIS (Grant No. 1618814) and IIBR (Grant No. 2029552)
- NIH NIAID (Grant No. HG001392 and R01AI14181)
- NSERC Discovery Grant (Grant No. RGPIN-07185-2020)

GitHub link: <https://github.com/davidecenzato/PFP-eBWT>

contact: davide.cenzato@univr.it



Thank you for your attention