Faster Wavelet Tree Queries

Data Compression Conference (DCC 2024)

Matteo Ceregini, *Florian Kurpicz*, and Rossano Venturini

Operations on Sequences

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
a	c	c	e	s	s	a	n	d	s	e	l	e	c	t

Applications

- compression
- computational geometry
- pattern matching
- ...

Operations on Sequences

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
a	c	c	e	s	s	a	n	d	s	e	l	e	c	t

$\mathrm{rank}_s(10) = 3$

Applications

- compression
- computational geometry
- pattern matching
- ...

Operations on Sequences

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
a	c	c	e	s	s	a	n	d	s	e	l	e	c	t

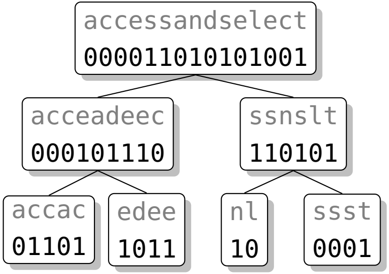
$\mathrm{rank}_s(10) = 3$

$\mathrm{select}_s(3) = 9$

Applications

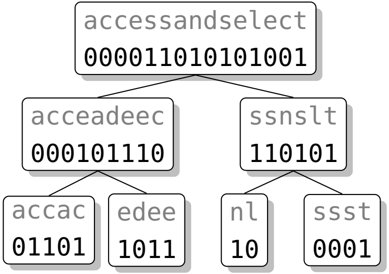
- compression
- computational geometry
- pattern matching
- ...

Wavelet Trees [GGV03]



- de-facto standard for access, rank, and queries
- $O(\log \sigma)$ query time
- require $\lceil H_0(T) \rceil n(1 + o(1))$ bits of space

Wavelet Trees [GGV03]



- de-facto standard for access, rank, and queries
- $O(\log\sigma)$ query time
- require $\lceil H_0(T)\rceil n(1+o(1))$ bits of space

Previous Work

- lots of work on construction [Bab+15; CNP15; DFK20; Din+21; Din+23; EK19; Fue+17; Kan18; LSB17; MNV16; Shu20]
- little work on queries [CNP15; Fer+07]

Faster Queries @ DCC'24

- "Faster Wavelet Tree Queries " (this paper)
- "Another Virtue of Wavelet Forests" (poster)

Wavelet ~~Trees~~ Matrices [CNP15]

```
accessandselect
000011010101001
```

$Z[0] = 9$

```
acceadeecssnslt
000101110110101
```

$Z[1] = 7$

```
accacnledeessst
011011010110001
```

$Z[2] = 7$

- alternative representation of wavelet trees
- “everything” known for trees applies to matrices

Wavelet ~~Trees~~ Matrices [CNP15]


```
accessandselect
000011010101001
```
$Z[0] = 9$

```
acceadeecssnslt
000101110110101
```
$Z[1] = 7$

```
accacnledeessst
011011010110001
```
$Z[2] = 7$

- alternative representation of wavelet trees
- “everything” known for trees applies to matrices

Construction

- bit vector on each level
- on *k*-th level symbols represented by *k*-th MSB
- stably sort sequence using written bit as key
- continue with next level
- store number of zeros on each level in Z

Rank Queries

`accessandselect`
`000011010101001`

$Z[0] = 9$

`acceadeecssnslt`
`000101110110101`

$Z[1] = 7$

`accacnledeessst`
`011011010110001`

$Z[2] = 7$

rank$_\alpha$*(i)*

$r_0 = i,\ b_0 = 0$

for $k = 0, \ldots, \ell$ **do**

$\alpha_k = (\alpha \ >> \ (\ell - 1 - k))\ \&\ 1$

$\texttt{offset} = \alpha_k * Z[k]$

$b_{k+1} = bv[k].rank_{\alpha_k}(b_k) + \texttt{offset}$

$r_{k+1} = bv[k].rank_{\alpha_k}(r_k) + \texttt{offset}$

return $r_{\ell+1} - b_{\ell+1}$

Rank Queries

```
accessandselect
000011010101001
```

$Z[0] = 9$

```
acceadeecssnslt
000101110110101
```

$Z[1] = 7$

```
accacnledeessst
011011010110001
```

$Z[2] = 7$

rank$_\alpha$*(i)*

$r_0 = i,\ b_0 = 0$

for $k = 0, \ldots, \ell$ **do**

$\alpha_k = (\alpha\ >>\ (\ell - 1 - k))\ \&\ 1$

$\texttt{offset} = \alpha_k * Z[k]$

$b_{k+1} = bv[k].rank_{\alpha_k}(b_k) + \texttt{offset}$

$r_{k+1} = bv[k].rank_{\alpha_k}(r_k) + \texttt{offset}$

return $r_{\ell+1} - b_{\ell+1}$

Cache Misses on Each Level

- binary rank and select queries are expensive
- rank/select data structures not in cache

4-Ary Wavelet Matrices

- use quad vectors instead of bit vectors
- space overhead 3.51 % $\rightsquigarrow$ 6.25 %
- rank possible with 2.41 % space overhead

- $\lceil \log \sigma / 2 \rceil$ levels (uncompressed)
- halve cache misses for rank/select data structures

```
accessandselect
000011010101001
000111001110101
```

$C_0 = [0, 5, 9, 11]$

```
accacnledeessst
011011010110001
```

$C_1 = [0, 7]$

$C_i[\alpha]$ number elements $< \alpha$ in level i

4-Ary Wavelet Matrices

- use quad vectors instead of bit vectors
- space overhead 3.51 % $\rightsquigarrow$ 6.25 %
- rank possible with 2.41 % space overhead

- $\lceil \log \sigma / 2 \rceil$ levels (uncompressed)
- halve cache misses for rank/select data structures

- more for rank queries in the tree/matrix
- path through tree/matrix known at query time

```
accessandselect
000011010101001
000111001110101

accacnledeessst
011011010110001
```

$C_0 = [0, 5, 9, 11]$

$C_1 = [0, 7]$

$C_i[\alpha]$ number elements $< \alpha$ in level i

Rank with Additive Approximation (RAA)

Karlsruhe Institute of Technology

Definition. Let $Q[1, n]$ be a quad vector and $\epsilon \in \mathbb{N}$.
The RAA for a position *i* and a symbol $\alpha \in [0, 3]$ is

$$rank_\alpha(i) \in [rank^{\approx}_\alpha(i), rank^{\approx}_\alpha(i) + \epsilon).$$

Rank with Additive Approximation (RAA)

Definition. Let $Q[1, n]$ be a quad vector and $\epsilon \in \mathbb{N}$.
The RAA for a position i and a symbol $\alpha \in [0, 3]$ is

$$rank_\alpha(i) \in [rank^{\approx}_\alpha(i), rank^{\approx}_\alpha(i) + \epsilon).$$

Lemma. The RAA for quad vectors can be solved in constant time using $\Theta(n/\epsilon)$ bits of space.

Rank with Additive Approximation (RAA)

Definition. Let $Q[1,n]$ be a quad vector and $\epsilon \in \mathbb{N}$. The RAA for a position i and a symbol $\alpha \in [0,3]$ is

$$rank_\alpha(i) \in [rank_\alpha^{\approx}(i), rank_\alpha^{\approx}(i) + \epsilon).$$

Lemma. The RAA for quad vectors can be solved in constant time using $\Theta(n/\epsilon)$ bits of space.

Proof (Sketch). Split quad vector $Q[1..n]$ into blocks of size $\epsilon/2$.

Q


Rank with Additive Approximation (RAA)

Definition. Let $Q[1, n]$ be a quad vector and $\epsilon \in \mathbb{N}$. The RAA for a position i and a symbol $\alpha \in [0, 3]$ is

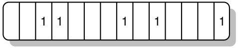
$$rank_\alpha(i) \in [rank_\alpha^{\approx}(i), rank_\alpha^{\approx}(i) + \epsilon).$$

Lemma. The RAA for quad vectors can be solved in constant time using $\Theta(n/\epsilon)$ bits of space.

Proof (Sketch). Split quad vector $Q[1..n]$ into blocks of size $\epsilon/2$.

Q 

Bit vectors $B_\alpha[1..\lceil 2n/\epsilon \rceil]$ mark blocks containing positions with $rank_\alpha(i) = 0 \mod \epsilon/2$ for $\alpha \in [0, 3]$.

B_α 

Rank with Additive Approximation (RAA)

Karlsruhe Institute of Technology

Definition. Let $Q[1, n]$ be a quad vector and $\epsilon \in \mathbb{N}$. The RAA for a position i and a symbol $\alpha \in [0, 3]$ is

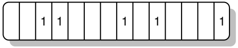
$$rank_\alpha(i) \in [rank_\alpha^{\approx}(i), rank_\alpha^{\approx}(i) + \epsilon).$$

Lemma. The RAA for quad vectors can be solved in constant time using $\Theta(n/\epsilon)$ bits of space.

Proof (Sketch). Split quad vector $Q[1..n]$ into blocks of size $\epsilon/2$.

Q


Bit vectors $B_\alpha[1..\lceil 2n/\epsilon \rceil]$ mark blocks containing positions with $rank_\alpha(i) = 0 \mod \epsilon/2$ for $\alpha \in [0, 3]$.

B_α


- to compute $rank_\alpha^{\approx}(i)$

Rank with Additive Approximation (RAA)

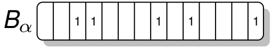
Definition. Let $Q[1,n]$ be a quad vector and $\epsilon \in \mathbb{N}$. The RAA for a position i and a symbol $\alpha \in [0,3]$ is

$$rank_\alpha(i) \in [rank_\alpha^{\approx}(i), rank_\alpha^{\approx}(i) + \epsilon).$$

Lemma. The RAA for quad vectors can be solved in constant time using $\Theta(n/\epsilon)$ bits of space.

Proof (Sketch). Split quad vector $Q[1..n]$ into blocks of size $\epsilon/2$.

Bit vectors $B_\alpha[1..\lceil 2n/\epsilon \rceil]$ mark blocks containing positions with $rank_\alpha(i) = 0 \mod \epsilon/2$ for $\alpha \in [0,3]$.



- to compute $rank_\alpha^{\approx}(i)$
- let $j = \lfloor 2i/\epsilon \rfloor$ and $k = B_\alpha.rank_1(j)$

Rank with Additive Approximation (RAA)

Definition. Let $Q[1, n]$ be a quad vector and $\epsilon \in \mathbb{N}$. The RAA for a position i and a symbol $\alpha \in [0, 3]$ is

$$rank_\alpha(i) \in [rank_\alpha^{\approx}(i), rank_\alpha^{\approx}(i) + \epsilon).$$

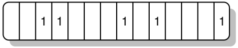
Lemma. The RAA for quad vectors can be solved in constant time using $\Theta(n/\epsilon)$ bits of space.

Proof (Sketch). Split quad vector $Q[1..n]$ into blocks of size $\epsilon/2$.

Q

Bit vectors $B_\alpha[1..\lceil 2n/\epsilon \rceil]$ mark blocks containing positions with $rank_\alpha(i) = 0 \mod \epsilon/2$ for $\alpha \in [0, 3]$.

B_α



- to compute $rank_\alpha^{\approx}(i)$
- let $j = \lfloor 2i/\epsilon \rfloor$ and $k = B_\alpha.rank_1(j)$
- $rank_\alpha(j \cdot \epsilon/2) \in [k \cdot \epsilon/2, k \cdot \epsilon/2 + \epsilon/2)$

Rank with Additive Approximation (RAA)

Definition. Let $Q[1, n]$ be a quad vector and $\epsilon \in \mathbb{N}$. The RAA for a position i and a symbol $\alpha \in [0, 3]$ is

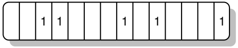
$$rank_\alpha(i) \in [rank^{\approx}_\alpha(i), rank^{\approx}_\alpha(i) + \epsilon).$$

Lemma. The RAA for quad vectors can be solved in constant time using $\Theta(n/\epsilon)$ bits of space.

Proof (Sketch). Split quad vector $Q[1..n]$ into blocks of size $\epsilon/2$.

Q 

Bit vectors $B_\alpha[1..\lceil 2n/\epsilon \rceil]$ mark blocks containing positions with $rank_\alpha(i) = 0 \mod \epsilon/2$ for $\alpha \in [0, 3]$.

B_α 

- to compute $rank^{\approx}_\alpha(i)$
- let $j = \lfloor 2i/\epsilon \rfloor$ and $k = B_\alpha.rank_1(j)$
- $rank_\alpha(j \cdot \epsilon/2) \in [k \cdot \epsilon/2, k \cdot \epsilon/2 + \epsilon/2)$
- $rank_\alpha(i) \in [k \cdot \epsilon/2, k \cdot \epsilon/2 + \epsilon)$

Rank with Additive Approximation (RAA)

Definition. Let $Q[1,n]$ be a quad vector and $\epsilon \in \mathbb{N}$. The RAA for a position i and a symbol $\alpha \in [0,3]$ is

$$rank_\alpha(i) \in [rank_\alpha^{\approx}(i), rank_\alpha^{\approx}(i) + \epsilon).$$

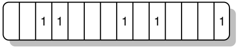
Lemma. The RAA for quad vectors can be solved in constant time using $\Theta(n/\epsilon)$ bits of space.

Proof (Sketch). Split quad vector $Q[1..n]$ into blocks of size $\epsilon/2$.

Q

Bit vectors $B_\alpha[1..\lceil 2n/\epsilon \rceil]$ mark blocks containing positions with $rank_\alpha(i) = 0 \mod \epsilon/2$ for $\alpha \in [0,3]$.

B_α



- to compute $rank_\alpha^{\approx}(i)$
- let $j = \lfloor 2i/\epsilon \rfloor$ and $k = B_\alpha.rank_1(j)$
- $rank_\alpha(j \cdot \epsilon/2) \in [k \cdot \epsilon/2, k \cdot \epsilon/2 + \epsilon/2)$
- $rank_\alpha(i) \in [k \cdot \epsilon/2, k \cdot \epsilon/2 + \epsilon)$
- $rank_\alpha^{\approx}(i) = k \cdot \epsilon/2$

Rank with Additive Approximation (RAA)

Definition. Let $Q[1, n]$ be a quad vector and $\epsilon \in \mathbb{N}$. The RAA for a position i and a symbol $\alpha \in [0, 3]$ is

$$rank_\alpha(i) \in [rank^{\approx}_\alpha(i), rank^{\approx}_\alpha(i) + \epsilon).$$

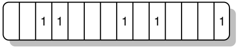
Lemma. The RAA for quad vectors can be solved in constant time using $\Theta(n/\epsilon)$ bits of space.

Lemma. A RAA data structure for quad vectors requires $\Omega(n/\epsilon)$ bits of space.

Proof (Sketch). Split quad vector $Q[1..n]$ into blocks of size $\epsilon/2$.

Q 

Bit vectors $B_\alpha[1..\lceil 2n/\epsilon \rceil]$ mark blocks containing positions with $rank_\alpha(i) = 0 \mod \epsilon/2$ for $\alpha \in [0, 3]$.

B_α 

- to compute $rank^{\approx}_\alpha(i)$
- let $j = \lfloor 2i/\epsilon \rfloor$ and $k = B_\alpha.rank_1(j)$
- $rank_\alpha(j \cdot \epsilon/2) \in [k \cdot \epsilon/2, k \cdot \epsilon/2 + \epsilon/2)$
- $rank_\alpha(i) \in [k \cdot \epsilon/2, k \cdot \epsilon/2 + \epsilon)$
- $rank^{\approx}_\alpha(i) = k \cdot \epsilon/2$

Predicting Cache Lines in the Wavelet Tree

Problem

- let r_k be the rank on the k-th level
- RAA does not guarantee that $r_k \in [r_k^{\approx}, r_k^{\approx} + \epsilon)$
- we can only compute $rank_{\alpha_k}^{\approx}(r_{k-1}^{\approx})$
- error could be up to $(k-1)\epsilon$

Predicting Cache Lines in the Wavelet Tree

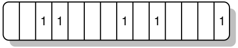
Karlsruhe Institute of Technology

Problem

- let r_k be the rank on the k-th level
- RAA does not guarantee that $r_k \in [r_k^{\approx}, r_k^{\approx} + \epsilon)$
- we can only compute $rank_{\alpha_k}^{\approx}(r_{k-1}^{\approx})$
- error could be up to $(k-1)\epsilon$

Solution

- store position where $rank_\alpha(i) = 0 \mod \epsilon/2$

B_α 

- $O(\log \epsilon)$ bits per position (offset)
- store in $D_{k,\alpha}$
- use rank/select on bit vector to access offset

Predicting Cache Lines in the Wavelet Tree

Problem

- let r_k be the rank on the k-th level
- RAA does not guarantee that $r_k \in [r_k^{\approx}, r_k^{\approx} + \epsilon)$
- we can only compute $rank_{\alpha_k}^{\approx}(r_{k-1}^{\approx})$
- error could be up to $(k-1)\epsilon$

Solution

- store position where $rank_\alpha(i) = 0 \mod \epsilon/2$

B_α			1	1				1		1				1

- $O(\log \epsilon)$ bits per position (offset)
- store in $D_{k,\alpha}$
- use rank/select on bit vector to access offset

- let d_{k-1} be successor of r_{k-1} in D_{k,α_k}
- $\Delta = \min(d_{k-1} - r_{k-1}^{\approx}, \epsilon - 1)$
- $r_k^{\approx} = rank_{\alpha_k}(d_{k-1}) - \Delta$

Predicting Cache Lines in the Wavelet Tree

Problem

- let r_k be the rank on the k-th level
- RAA does not guarantee that $r_k \in [r_k^{\approx}, r_k^{\approx} + \epsilon)$
- we can only compute $rank_{\alpha_k}^{\approx}(r_{k-1}^{\approx})$
- error could be up to $(k-1)\epsilon$

Solution

- store position where $rank_\alpha(i) = 0 \mod \epsilon/2$

B_α

		1	1				1		1				1

- $O(\log \epsilon)$ bits per position (offset)
- store in $D_{k,\alpha}$
- use rank/select on bit vector to access offset

- let d_{k-1} be successor of r_{k-1} in D_{k,α_k}
- $\Delta = \min(d_{k-1} - r_{k-1}^{\approx}, \epsilon - 1)$
- $r_k^{\approx} = rank_{\alpha_k}(d_{k-1}) - \Delta$

Lemma. At any level k, we have $r_k \in [r_k^{\approx}, r_k^{\approx} + \epsilon)$.

Predicting Cache Lines in the Wavelet Tree

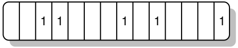
Karlsruhe Institute of Technology

Problem

- let r_k be the rank on the k-th level
- RAA does not guarantee that $r_k \in [r_k^{\approx}, r_k^{\approx} + \epsilon)$
- we can only compute $rank_{\alpha_k}^{\approx}(r_{k-1}^{\approx})$
- error could be up to $(k-1)\epsilon$

Solution

- store position where $rank_\alpha(i) = 0 \mod \epsilon/2$

B_α 

- $O(\log \epsilon)$ bits per position (offset)
- store in $D_{k,\alpha}$
- use rank/select on bit vector to access offset

- let d_{k-1} be successor of r_{k-1} in D_{k,α_k}
- $\Delta = \min(d_{k-1} - r_{k-1}^{\approx}, \epsilon - 1)$
- $r_k^{\approx} = rank_{\alpha_k}(d_{k-1}) - \Delta$

Lemma. At any level k, we have $r_k \in [r_k^{\approx}, r_k^{\approx} + \epsilon)$.

- requires $\Theta((n/\epsilon \log \sigma) \log \epsilon)$ bits of space
- problematic if predictor does not fit into cache
- use hierarchy of predictors

Practical Implementations and Experiments

- Δ and $D_{k,\alpha}$ not necessary
- error always small enough for σ up to 256
- prefetch more cache lines

- use two levels of predictors
- first level with $\epsilon = 2048$
- second level with $\epsilon = 256$
- help to prefetch blocks and super blocks

Practical Implementations and Experiments

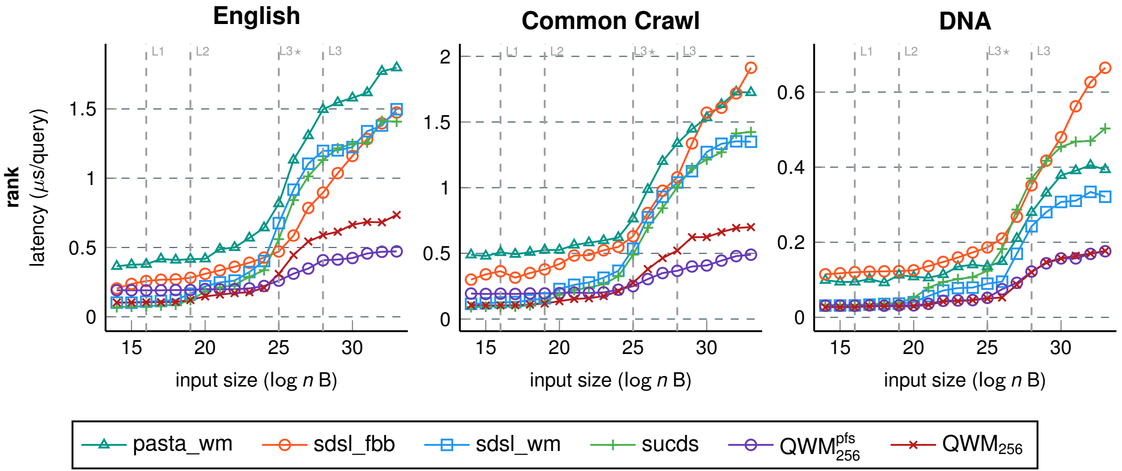
- Δ and $D_{k,\alpha}$ not necessary
- error always small enough for σ up to 256
- prefetch more cache lines

- use two levels of predictors
- first level with $\epsilon = 2048$
- second level with $\epsilon = 256$
- help to prefetch blocks and super blocks

Experimental Setup

- AMD EPYC 7713
  - 64 KB L1I and 64 KB L1D per core
  - 512 KB L2I+D per core
  - 256 MB L3I+D (32 MB per 8 cores CCX)
- 2 TB DDR4 RAM
- Ubuntu 20.04.3 LTS kernel version 5.4.0-155
- C++: GCC 11.1.0 (`-O3 -march=native`)
- Rust: `cargo build -release`

Experimental Evaluation (Latency)



Conclusion and Future Work

Karlsruhe Institute of Technology

This Paper

- up to 3 times faster wavelet tree queries
- predictive model for rank queries

What's Next

- compressed wavelet trees
- use predictive model for other data structures

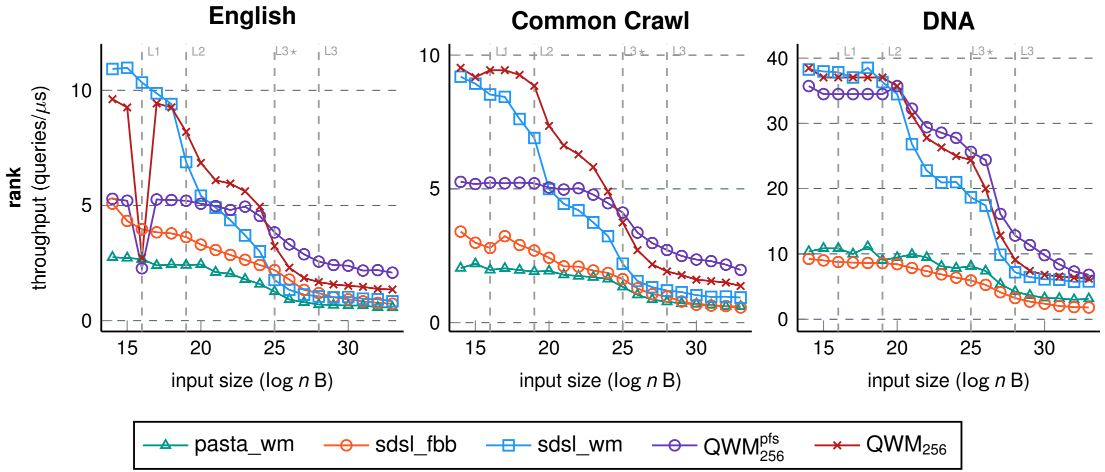
Check It Out

- `https://github.com/rossanoventurini/qwt`

European Research Council
Established by the European Commission

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 882500) and innovation programme (grant agreement No. 882500), by the PNRR ECS00000017 Tuscany Health Ecosystem Spoke 6 "Precision medicine & personalized healthcare", by the "Algorithms, Data Structures and Combinatorics for Machine Learning" (MIUR-PRIN 2017), and by the "Algorithmic Problems and Machine Learning" (MIUR-PRIN 2022).

Experimental Evaluation (Throughput)



Bibliography I

[Bab+15] Maxim A. Babenko, Pawel Gawrychowski, Tomasz Kociumaka, and Tatiana Starikovskaya. "Wavelet Trees Meet Suffix Trees". In: *SODA*. SIAM, 2015, pages 572–591. DOI: 10.1137/1.9781611973730.39.

[CNP15] Francisco Claude, Gonzalo Navarro, and Alberto Ordóñez Pereira. "The Wavelet Matrix: An Efficient Wavelet Tree for Large Alphabets". In: *Inf. Syst.* 47 (2015), pages 15–32. DOI: 10.1016/j.is.2014.06.002.

[DFK20] Patrick Dinklage, Johannes Fischer, and Florian Kurpicz. "Constructing the Wavelet Tree and Wavelet Matrix in Distributed Memory". In: *ALENEX*. SIAM, 2020, pages 214–228. DOI: 10.1137/1.9781611976007.17.

[Din+21] Patrick Dinklage, Jonas Ellert, Johannes Fischer, Florian Kurpicz, and Marvin Löbel. "Practical Wavelet Tree Construction". In: *ACM J. Exp. Algorithmics* 26 (2021), 1.8:1–1.8:67. DOI: 10.1145/3457197.

Bibliography II

[Din+23] Patrick Dinklage, Johannes Fischer, Florian Kurpicz, and Jan-Philipp Tarnowski. “Bit-Parallel (Compressed) Wavelet Tree Construction”. In: *DCC*. IEEE, 2023, pages 81–90. DOI: 10.1109/DCC55655.2023.00016.

[EK19] Jonas Ellert and Florian Kurpicz. “Parallel External Memory Wavelet Tree and Wavelet Matrix Construction”. In: *SPIRE*. Volume 11811. Lecture Notes in Computer Science. Springer, 2019, pages 392–406. DOI: 10.1007/978-3-030-32686-9_28.

[Fer+07] Paolo Ferragina, Giovanni Manzini, Veli Mäkinen, and Gonzalo Navarro. “Compressed representations of sequences and full-text indexes”. In: *ACM Trans. Algorithms* 3.2 (2007), page 20. DOI: 10.1145/1240233.1240243.

[Fue+17] José Fuentes-Sepúlveda, Erick Elejalde, Leo Ferres, and Diego Seco. “Parallel construction of wavelet trees on multicore architectures”. In: *Knowl. Inf. Syst.* 51.3 (2017), pages 1043–1066.

[GGV03] Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. “High-Order Entropy-Compressed Text Indexes”. In: *SODA*. ACM/SIAM, 2003, pages 841–850.

Bibliography III

[Kan18] Yusaku Kaneta. “Fast Wavelet Tree Construction in Practice”. In: *SPIRE*. Volume 11147. Lecture Notes in Computer Science. Springer, 2018, pages 218–232. DOI: 10.1007/978-3-030-00479-8_18.

[LSB17] Julian Labeit, Julian Shun, and Guy E. Blelloch. “Parallel lightweight wavelet tree, suffix array and FM-index construction”. In: *J. Discrete Algorithms* 43 (2017), pages 2–17. DOI: 10.1016/j.jda.2017.04.001.

[MNV16] J. Ian Munro, Yakov Nekrich, and Jeffrey Scott Vitter. “Fast construction of wavelet trees”. In: *Theor. Comput. Sci.* 638 (2016), pages 91–97. DOI: 10.1016/j.tcs.2015.11.011.

[Shu20] Julian Shun. “Improved parallel construction of wavelet trees and rank/select structures”. In: *Inf. Comput.* 273 (2020), page 104516. DOI: 10.1016/j.ic.2020.104516.