PRIN “Multicriteria data structures”, 4th meeting

Repetition- and linearity-aware rank/select dictionaries

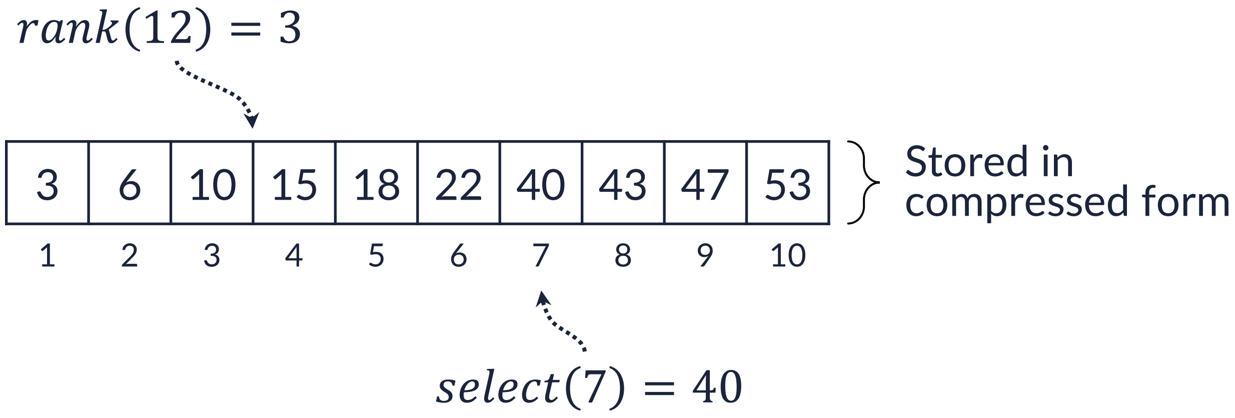
Paolo Ferragina Giovanni Manzini Giorgio Vinciguerra

Università di Pisa

(ISAAC 2021)

Compressed rank/select dictionaries

- Given a set A of n elements over an integer universe $0,1,\dots,u$
  1. Store them in compressed form
  2. Implement $rank(x)$: number of elements in A which are $\leq x$
  3. Implement $select(i)$: return the ith smallest element in A
- Well-studied building block of succinct data structures



$rank(12) = 3$

3	6	10	15	18	22	40	43	47	53
1	2	3	4	5	6	7	8	9	10

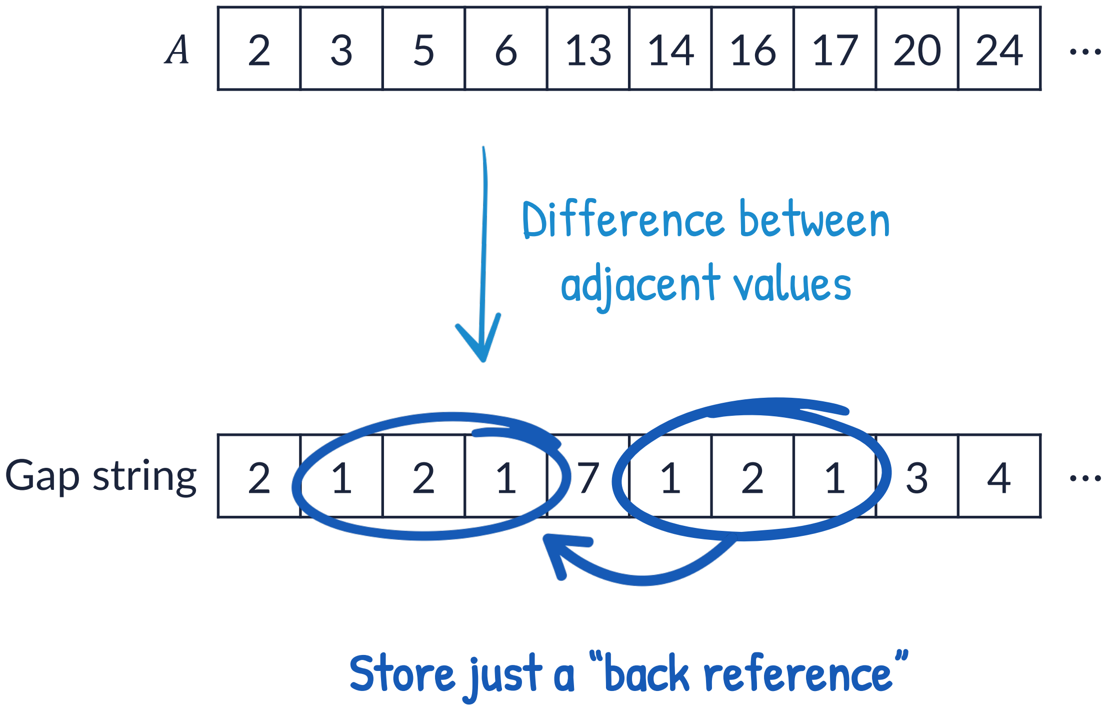
Stored in compressed form

$select(7) = 40$

Two sources of compressibility

We exploit them both

Repetitiveness



A										
2	3	5	6	13	14	16	17	20	24	…

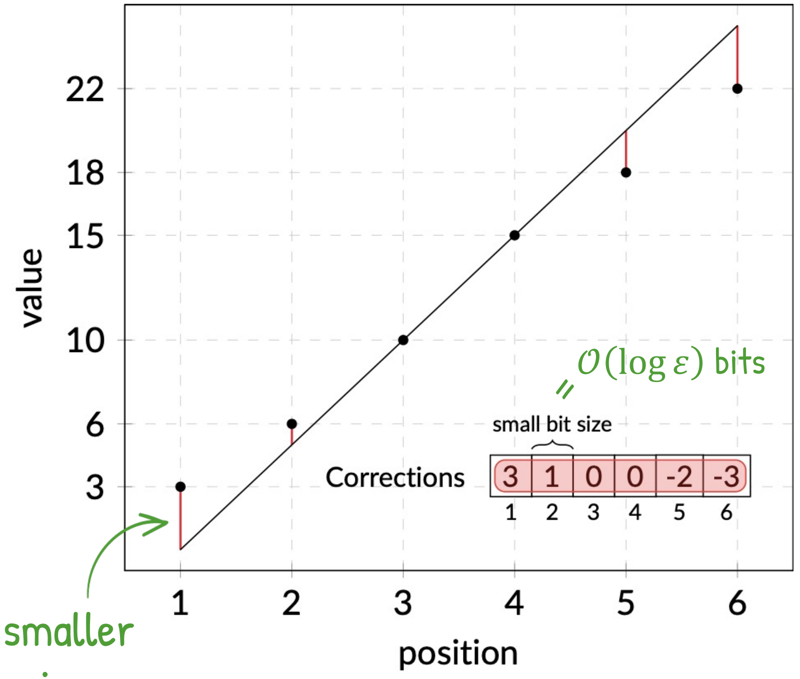
Difference between adjacent values

Gap string										
2	1	2	1	7	1	2	1	3	4	…

Store just a "back reference"

Approximate linearity

[Boffa et al., ALENEX '21]



Errors smaller than a given integer ε

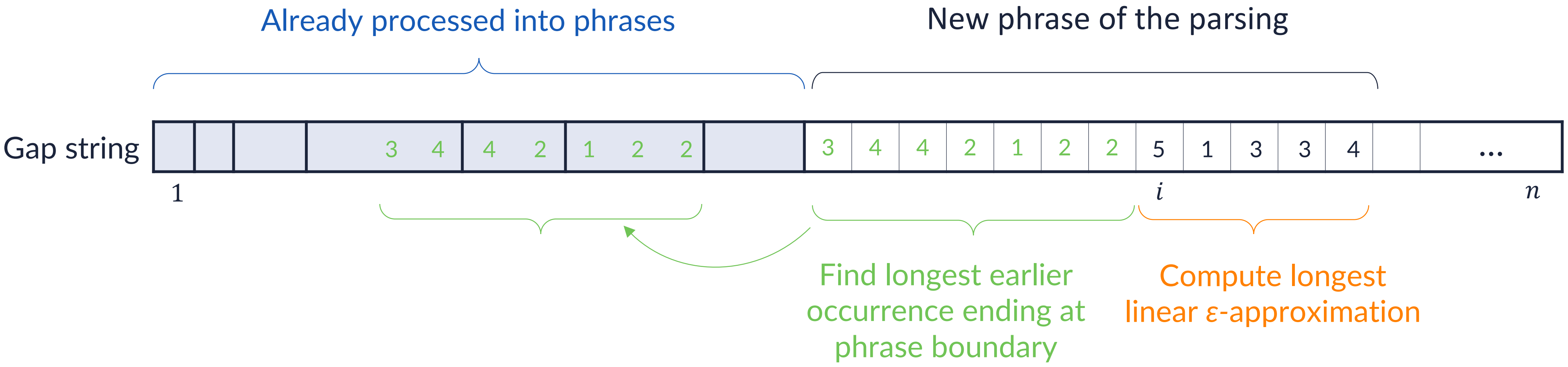
A					
3	6	10	15	18	22
1	2	3	4	5	6

Many nonlinear points
⇓
Use piecewise linear ε-approx.

Exploiting repetitiveness and approx. linearity

1. Build on two known repetition-aware methods
    - Lempel-Ziv parsing, LZ-End [Kreft and Navarro, TCS 2013]
    - Block tree [Belazzougui et al., JCSS 2021]
2. Augment them to use linear ε-approximations with corrections
3. Show how to support $rank$ and $select$ in space bounded by the high-order entropy or a repetitiveness measure of the gaps

The LZ$_\varepsilon$ parsing



Already processed into phrases

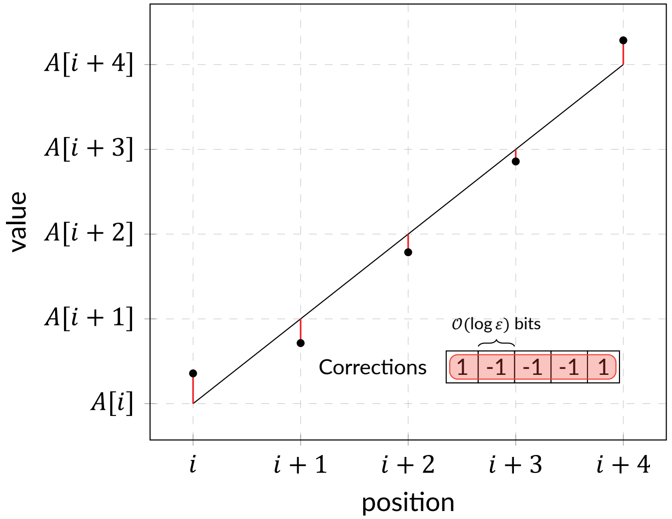
New phrase of the parsing

Gap string

			3 4	4 2	1 2 2		3	4	4	2	1	2	2	5	1	3	3	4		…

Find longest earlier occurrence ending at phrase boundary

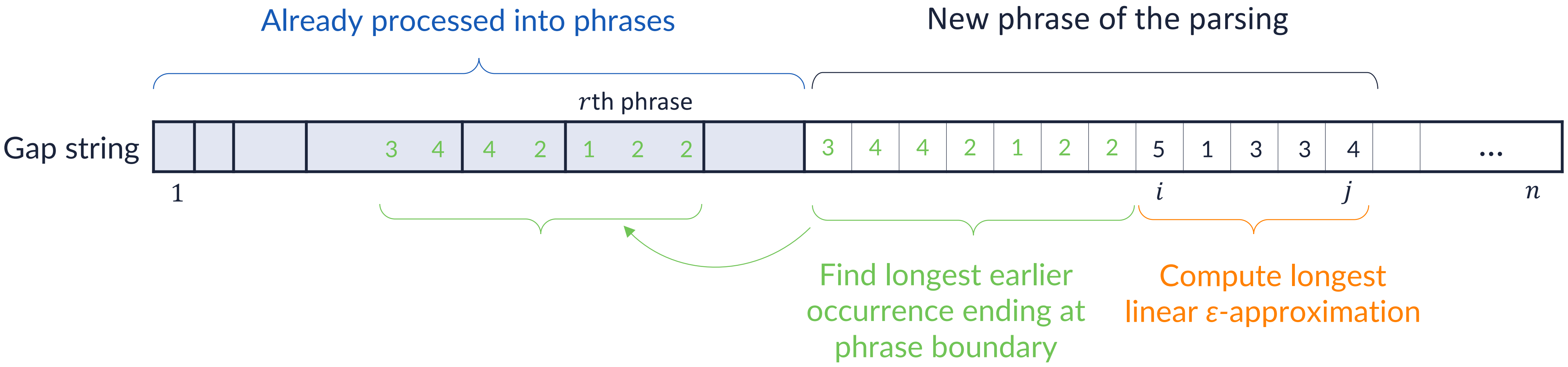
Compute longest linear ε-approximation



Corrections

1	-1	-1	-1	1

The LZ$_\varepsilon$ parsing



Already processed into phrases

New phrase of the parsing

rth phrase

Gap string

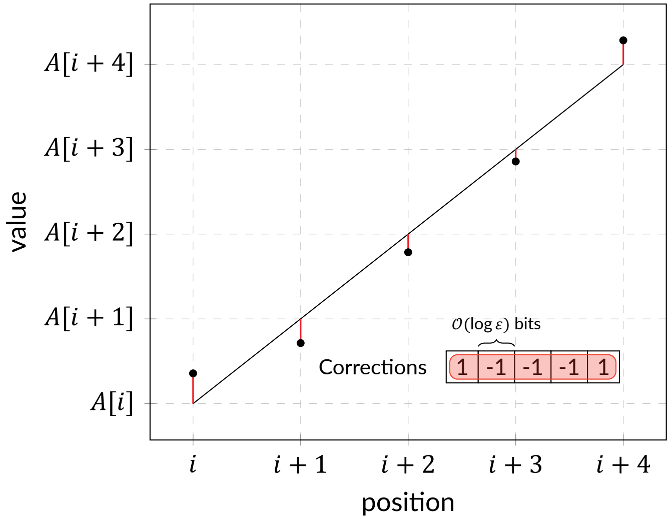
			3 4	4 2	1 2 2		3	4	4	2	1	2	2	5	1	3	3	4		…

Find longest earlier occurrence ending at phrase boundary

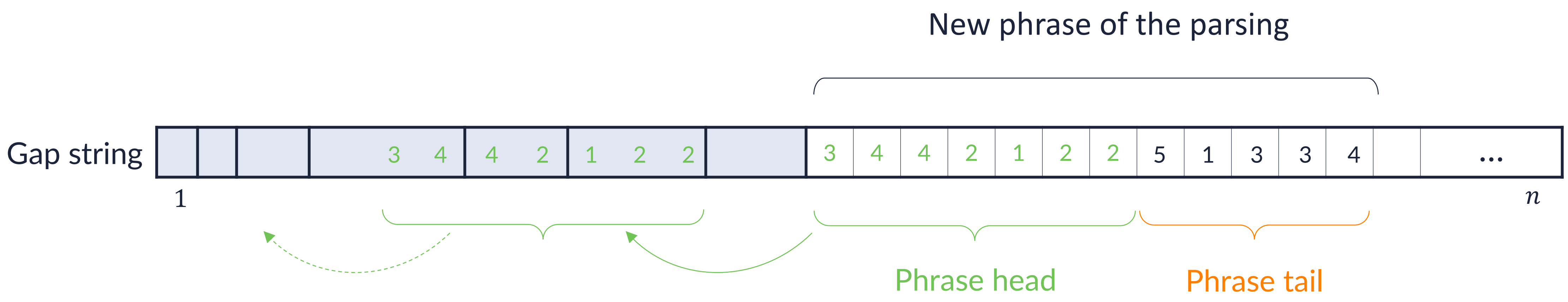
Compute longest linear ε-approximation

For the new phrase we store

- Indexes i, j, and r
- Slope and intercept of the line
- Array of $j - i + 1$ corrections, $\mathcal{O}(\log \varepsilon)$ bits each



Queries in the LZ_ε parsing



If $t \in$ Phrase head, then we must recursively unroll the source phrase

If $t \in$ Phrase tail, then we can answer directly with the linear ε-approx

$select(t)$
$rank(A[t])$

$\mathrm{LZ}_\varepsilon^\rho$: Introduce a trade-off parameter $\rho > 0$ to shorten the phrase head and make queries faster

$\mathrm{LZ}_\varepsilon^\rho$ bounds

→ No worse than a traditional LZ-parsing

→ No worse than LA-vector in space

Let $\sigma =$ number of distinct values in the gap string

Select time	$\mathcal{O}(\log^{1+\rho} n)$
Rank time	$\mathcal{O}(\log^{1+\rho} n + \log \varepsilon)$
Space in bits	$\underbrace{nH_k(\text{gap string}) + \mathcal{O}(n/\log^\rho n) + o(n \log \sigma)}_{\text{Exploit repetitions}} + \underbrace{\text{space for tails}}_{\text{Exploit approximate linearity}}$

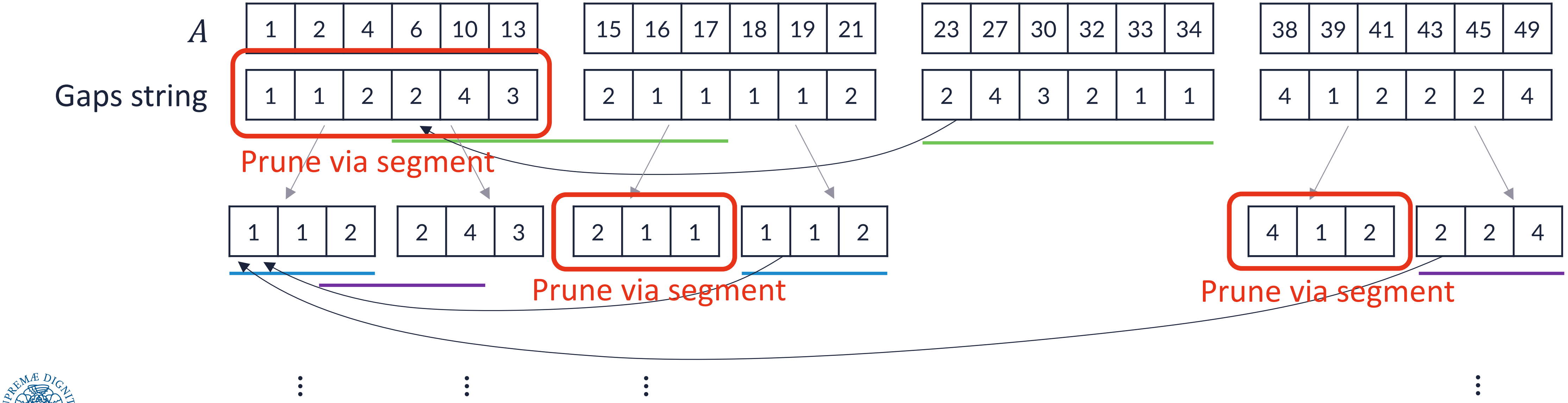
The block-ε tree

- Start with a standard block tree construction on the gap string

A	1	2	4	6	10	13	15	16	17	18	19	21	23	27	30	32	33	34	38	39	41	43	45	49
Gap string	1	1	2	2	4	3	2	1	1	1	1	2	2	4	3	2	1	1	4	1	2	2	2	4

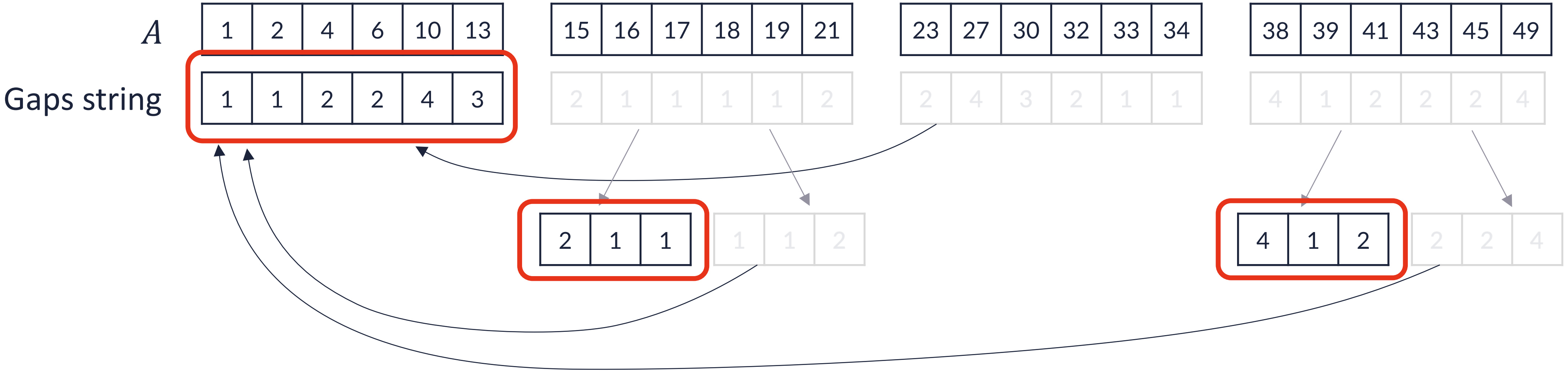
The block-ε tree

- Start with a standard block tree construction on the gap string
- Assign to each node the bit cost of encoding its subtree
- Prune subtrees that are better compressed by linear ε-approximations



The block-ε tree

- Start with a standard block tree construction on the gap string
- Assign to each node the bit cost of encoding its subtree
- Prune subtrees that are better compressed by linear ε-approximations
- Store topology, leaf linear ε-approx., and left pointers of copied blocks



A	1	2	4	6	10	13	15	16	17	18	19	21	23	27	30	32	33	34	38	39	41	43	45	49
Gaps string	1	1	2	2	4	3	2	1	1	1	1	2	2	4	3	2	1	1	4	1	2	2	2	4

2	1	1	1	1	2	4	1	2	2	2	4

Block-ε tree bounds

- Based on the δ repetitiveness measure on strings:[1,2,3]

  $\delta = \max\{d_k/k : k = 1, \dots, n\}$
  where d_k = number of distinct substrings of length k in the gap string

- Number of levels is $h = \mathcal{O}\left(\log \frac{n}{\delta}\right)$

Select time	$\mathcal{O}(h)$
Rank time	$\mathcal{O}\left(\log \log \frac{u}{\delta} + h + \log \varepsilon\right)$
Space in bits	$\mathcal{O}\left(\delta \log \frac{u}{\delta} \log u\right)$

[1] Raskhodnikova et al., Algorithmica (2013)
[2] Christiansen et al., TALG (2020)
[3] Kociumaka et al., LATIN '20

Experiments with the block-ε tree

- Compared with LA-vector, and a standard block tree
- Datasets: postings lists, positions of symbols in texts (DNA, URLs)
- LA-vector is 10.5× faster in select and 4.7× faster in rank than block tree, but no clear winner in space → Combination of repetitiveness and approximate-linearity makes sense
- Our block-ε tree:
    - wrt LA-vector, it is always slower in select and in rank
    - wrt block tree, it is 2.2× faster in select, either faster (1.3×) or slower (1.3×) in rank
    - has the best space in 2/12 datasets, and the second-best space in 7/12 datasets

} Block-ε tree achieves a good compromise by exploiting both regularities

Code available at github.com/gvinciguerra/BlockEpsilonTree

Conclusions

- Exploit both repetitiveness and approx. linearity in rank/select dictionaries
- $\mathrm{LZ}_\varepsilon^\rho$ parsing
  - Combine backward copies and linear ε-approximations
  - Space complexity bounded by the kth order entropy
- Block-ε tree
  - Optimise block tree by compressing areas with high approximate linearity
  - Space-time bounds based on the δ repetitiveness measure
  - Experimentally achieves a good compromise between block trees and LA-vectors
- Future work
  - Implement $\mathrm{LZ}_\varepsilon^\rho$
  - Relation of approximate linearity with other compressibility measures