A Tutorial Workshop on ML for Systems and Systems for ML @ BTW 2023

Advances in data-aware compressed-indexing schemes for integer and string keys

Giorgio Vinciguerra

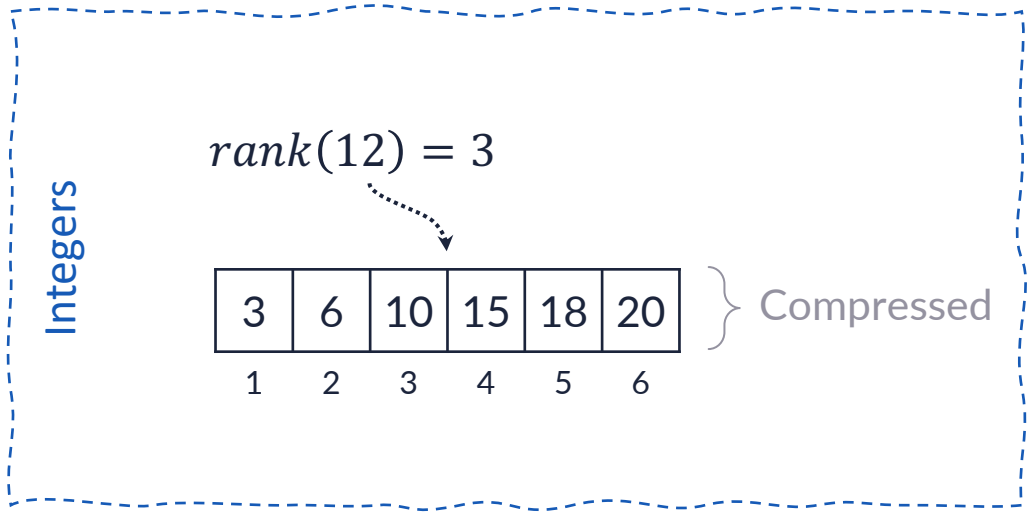
Università di Pisa

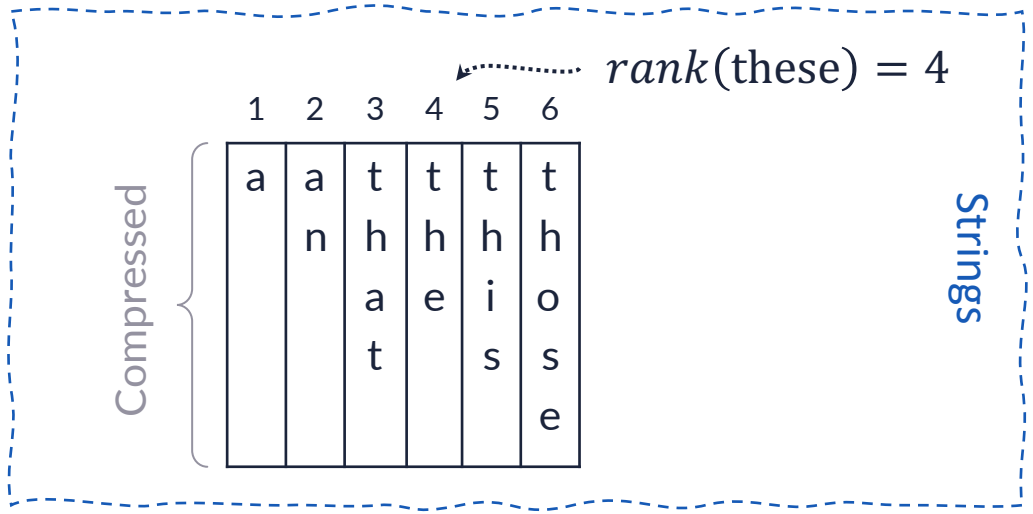
Based on papers with A. Boffa, P. Ferragina, G. Manzini, F. Tosoni

A compressed indexing problem

Harder than just indexing or just compressing

1. Store a sorted set A of n elements in compressed form
2. Implement random access
3. Implement $rank(x)$ = number of elements in A which are $\leq x$





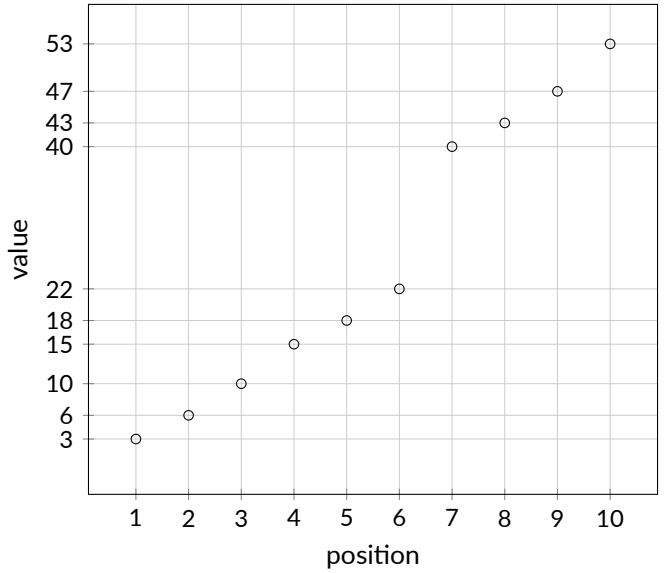
IP routing, succinct data structures, inverted indexes, ...

Query autocompletion, k-mer counting, range searches, ...

LA-vector: A learned approach for integers

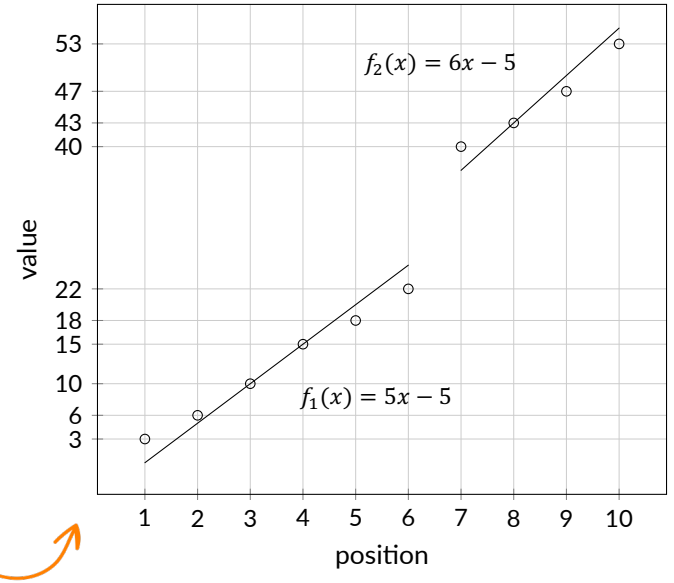
Compressed indexing via piecewise linear approximations (**PLAs**)

Step 1: map input to (pos, value)

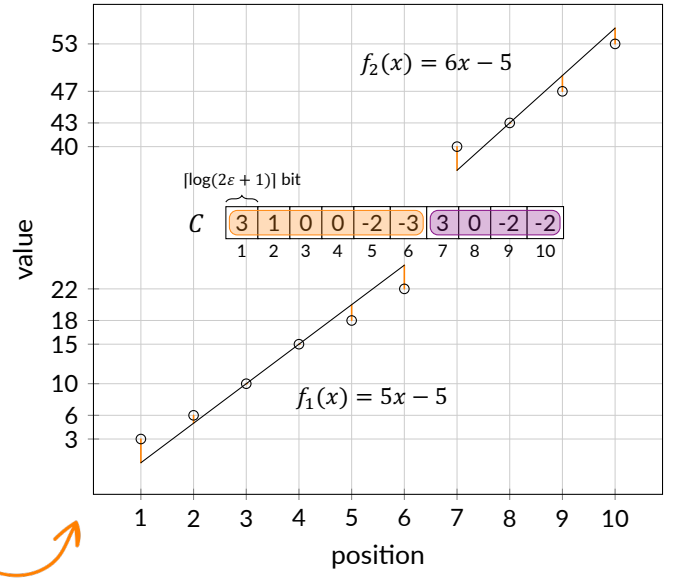


3	6	10	15	18	22	40	43	47	53
1	2	3	4	5	6	7	8	9	10

Step 2: build a PLA with max error ε

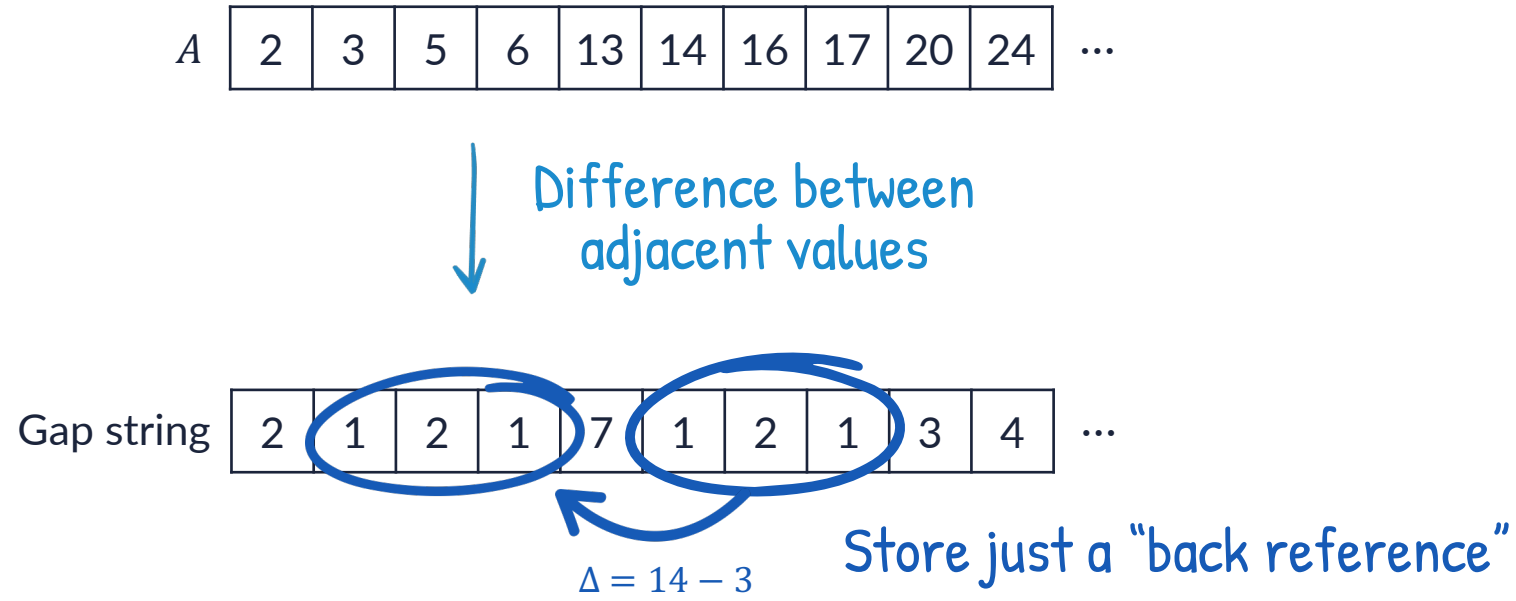


Step 3: store segments + corrections



Crucial ingredient: algorithm to learn a PLA that minimises the space by using different ε for different segments

What about other sources of compressibility?



LA-vector would store segments and corrections for the repeated sections!

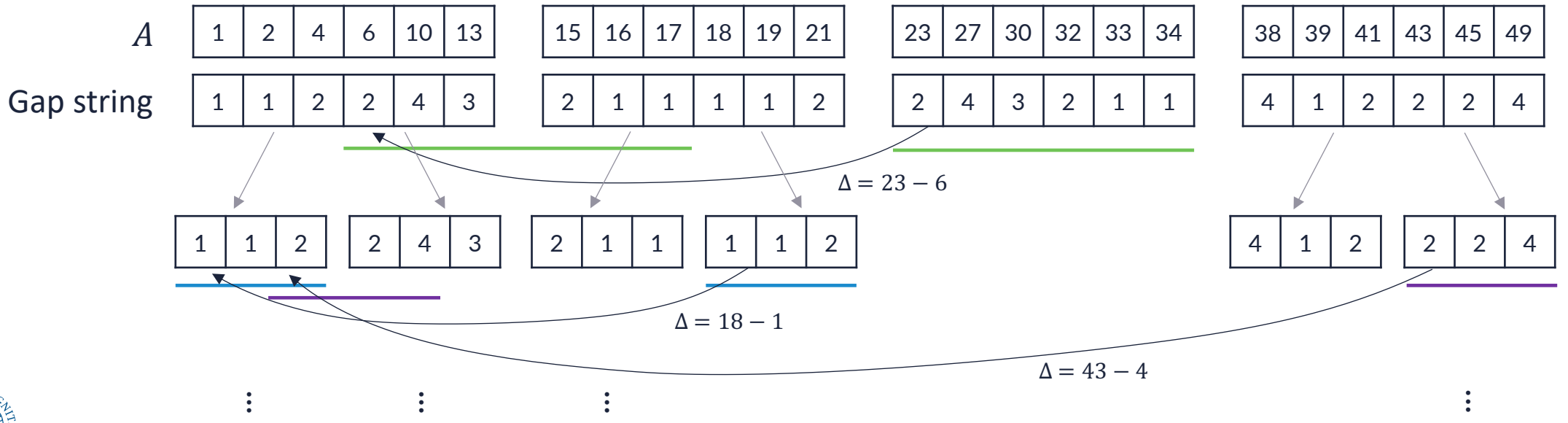
Block-ε tree: exploiting linearity and repetitiveness

- Split A and gap string into equal-sized blocks

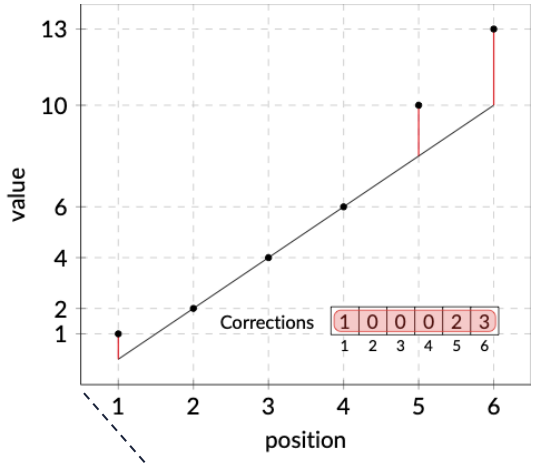
A	1	2	4	6	10	13	15	16	17	18	19	21	23	27	30	32	33	34	38	39	41	43	45	49
Gap string	1	1	2	2	4	3	2	1	1	1	1	2	2	4	3	2	1	1	4	1	2	2	2	4

Block-ε tree: exploiting linearity and repetitiveness

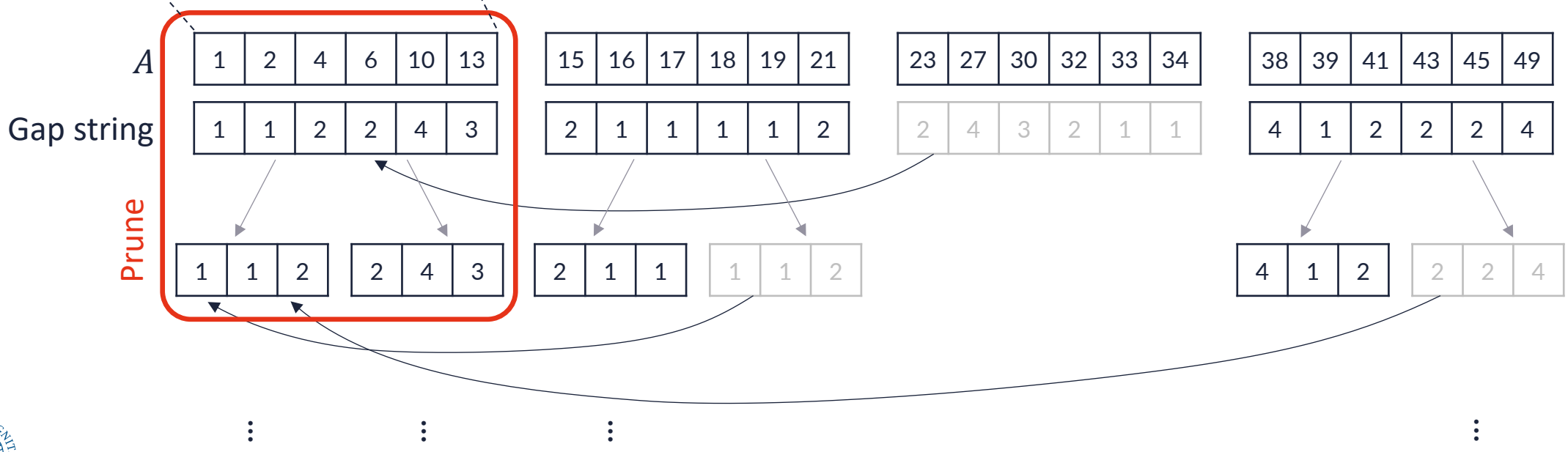
- Split A and gap string into equal-sized blocks
- Replace repeated blocks with a pointer to the first occurrence
- Split remaining blocks into two and repeat recursively
- For each block at any level, check if a learned model compresses better



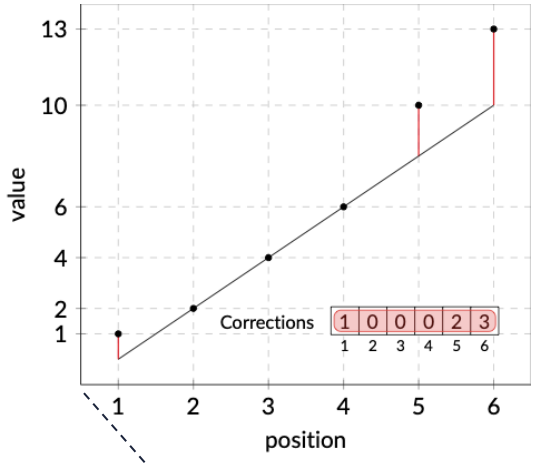
Block-ε tree: exploiting linearity and repetitiveness



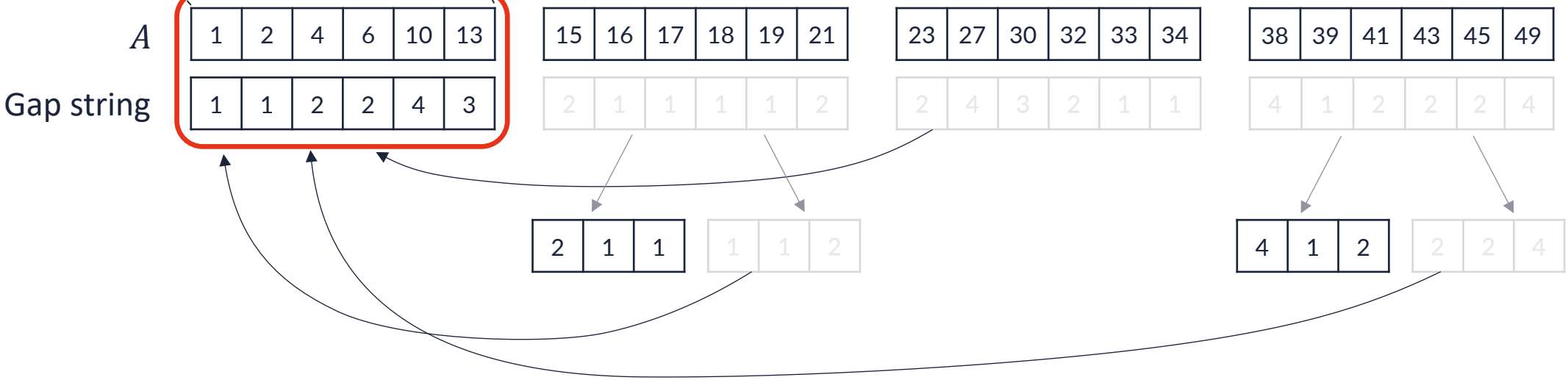
- Split A and gap string into equal-sized blocks
- Replace repeated blocks with a pointer to the first occurrence
- Split remaining blocks into two and repeat recursively
- For each block at any level, check if a learned model compresses better
  - If so, replace the block and the subtree with the model



Block-ε tree: exploiting linearity and repetitiveness



- Split A and gap string into equal-sized blocks
- Replace repeated blocks with a pointer to the first occurrence
- Split remaining blocks into two and repeat recursively
- For each block at any level, check if a learned model compresses better
  - If so, replace the block and the subtree with the model

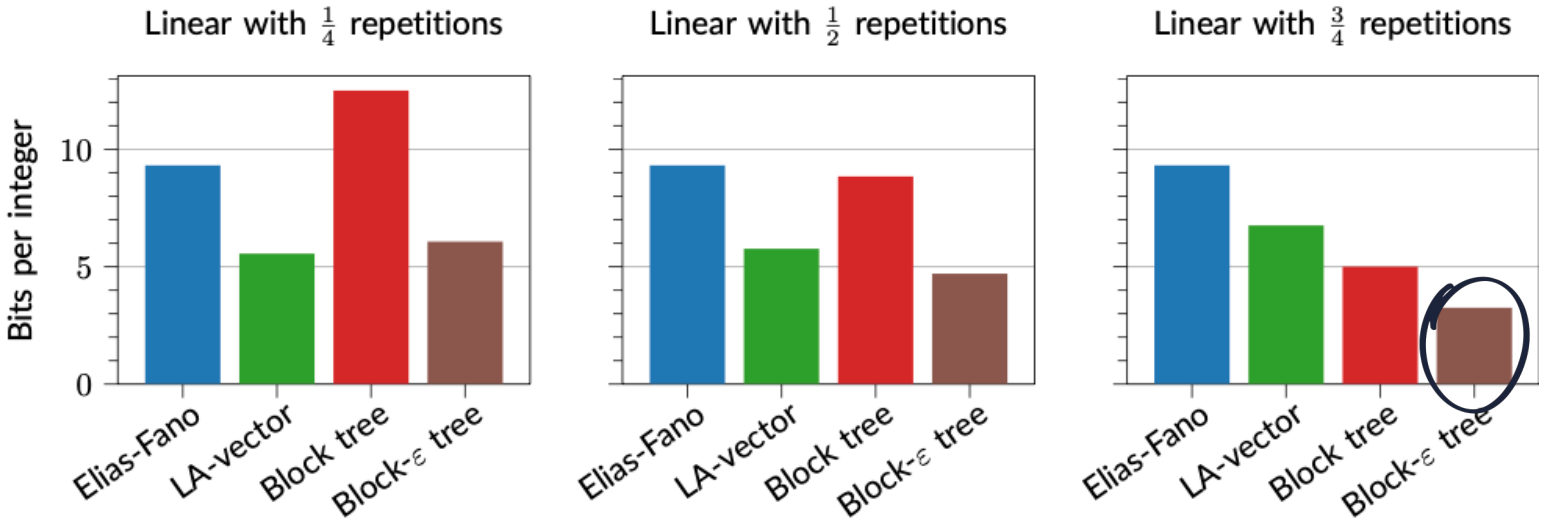


Experiments

On 11 standard datasets, up to 490M integers

- No clear winner in space between LA-vector (linearity-aware) and the block tree (repetition-aware)
- Block-ε tree achieves the best or the second-best space in the majority of datasets
- Block-ε tree has 5x slower random access and 4x slower rank than LA-vector

On datasets with explicit linearities and repetitions

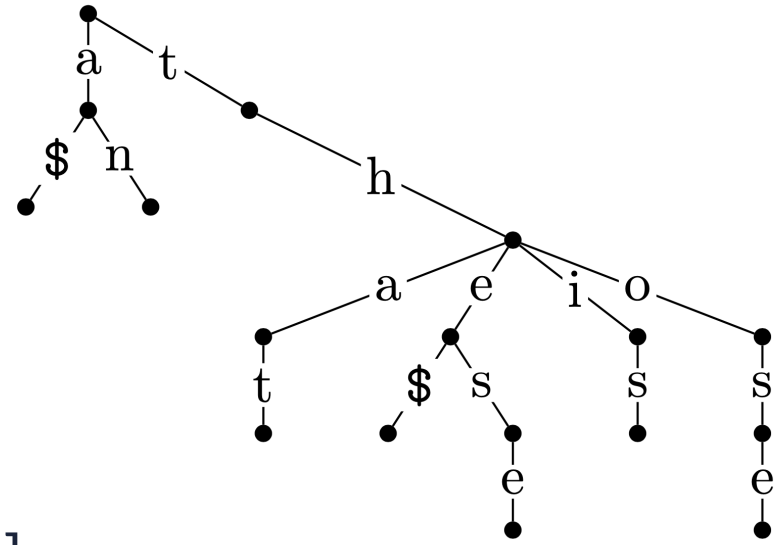


Block-ε tree exploits both repetitions and linearities

Compressed-indexing string keys

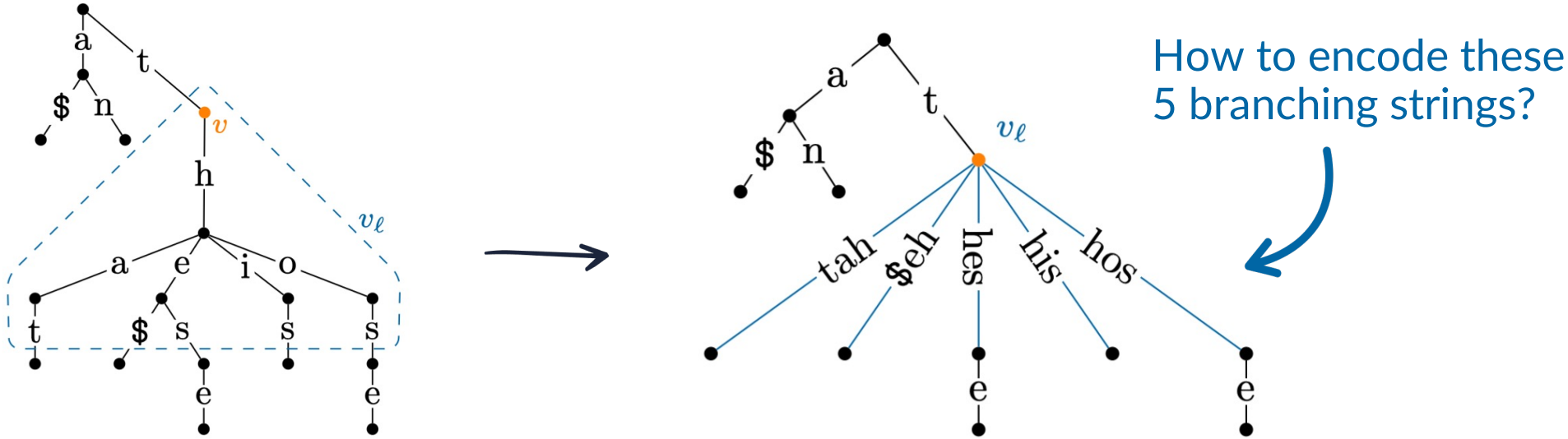
Compressed indexing of strings

- Tries are the classic solution
- Many improvements since the ’60s
  - Compact unary paths [J. ACM 68]
  - Adaptive node layouts [ICDE 13]
  - Cache-aware layouts [PODS 08]
  - Word packing [CPM 17]
  - Succinct topology representation [SIGMOD 18]
  - …
- **No practical solution is good on all different kinds of string data (URLs, k-mers, dictionary terms, …) → no data-awareness**



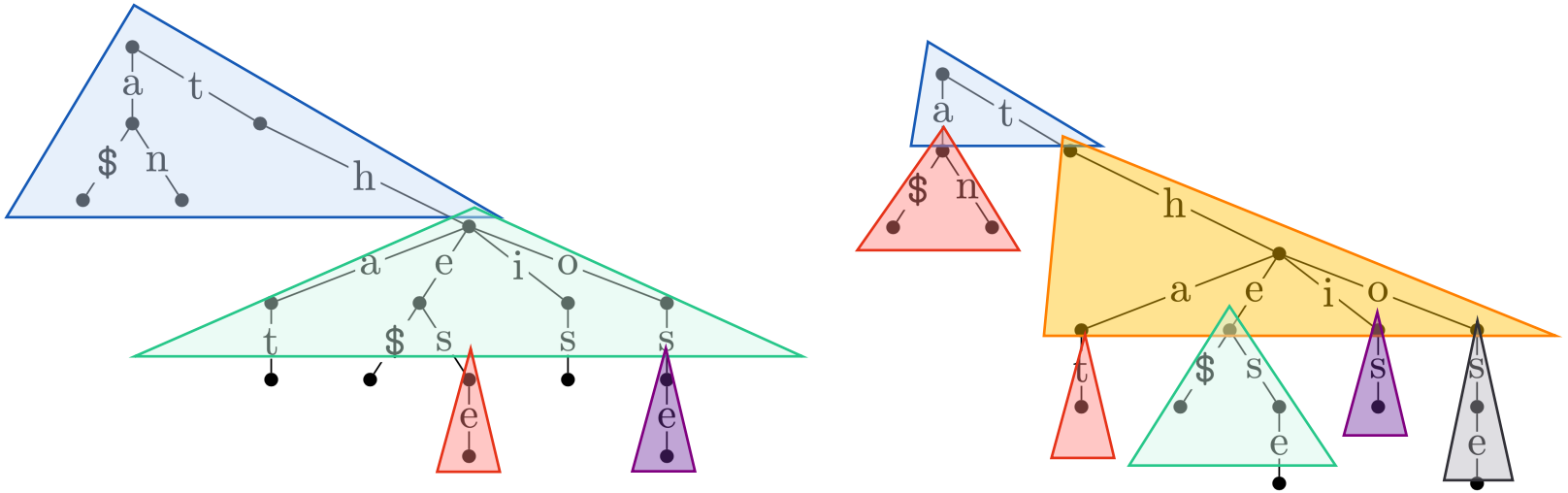
CoCo-trie: Compressed Collapsed trie

Key tool: collapsing subtries



1. Store the alphabet of the branching strings: $\{\$, a, e, h, i, o, s, t\}$, $\sigma = 8$ symbols in total
2. Map strings to ints in radix σ, e.g. hat $\rightarrow h\sigma^2 + a\sigma^1 + t\sigma^0 = 3 \cdot 8^2 + 1 \cdot 8^1 + 7 \cdot 8^0 = 207$
3. Keep the first string explicit (hat = 207)
4. Transform the rest to a list of differences:
   [he$–hat, hes–hat, his–hat, hos–hat] = [1, 7, 23, 31]
5. Apply a compressed indexing scheme for integers

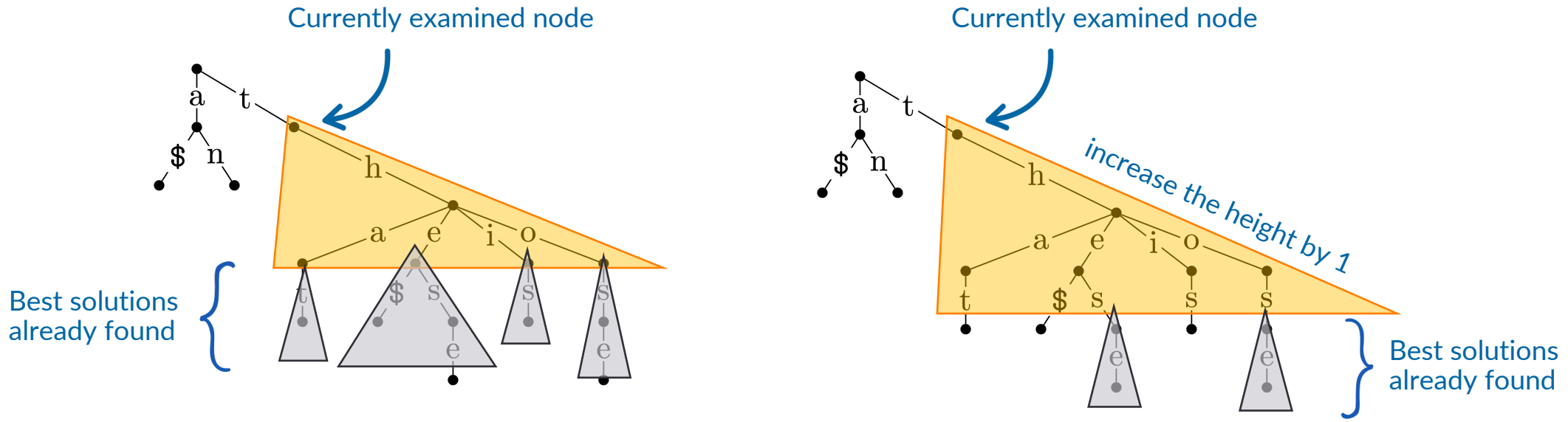
Which subtries to collapse?



... a large search space of different layouts

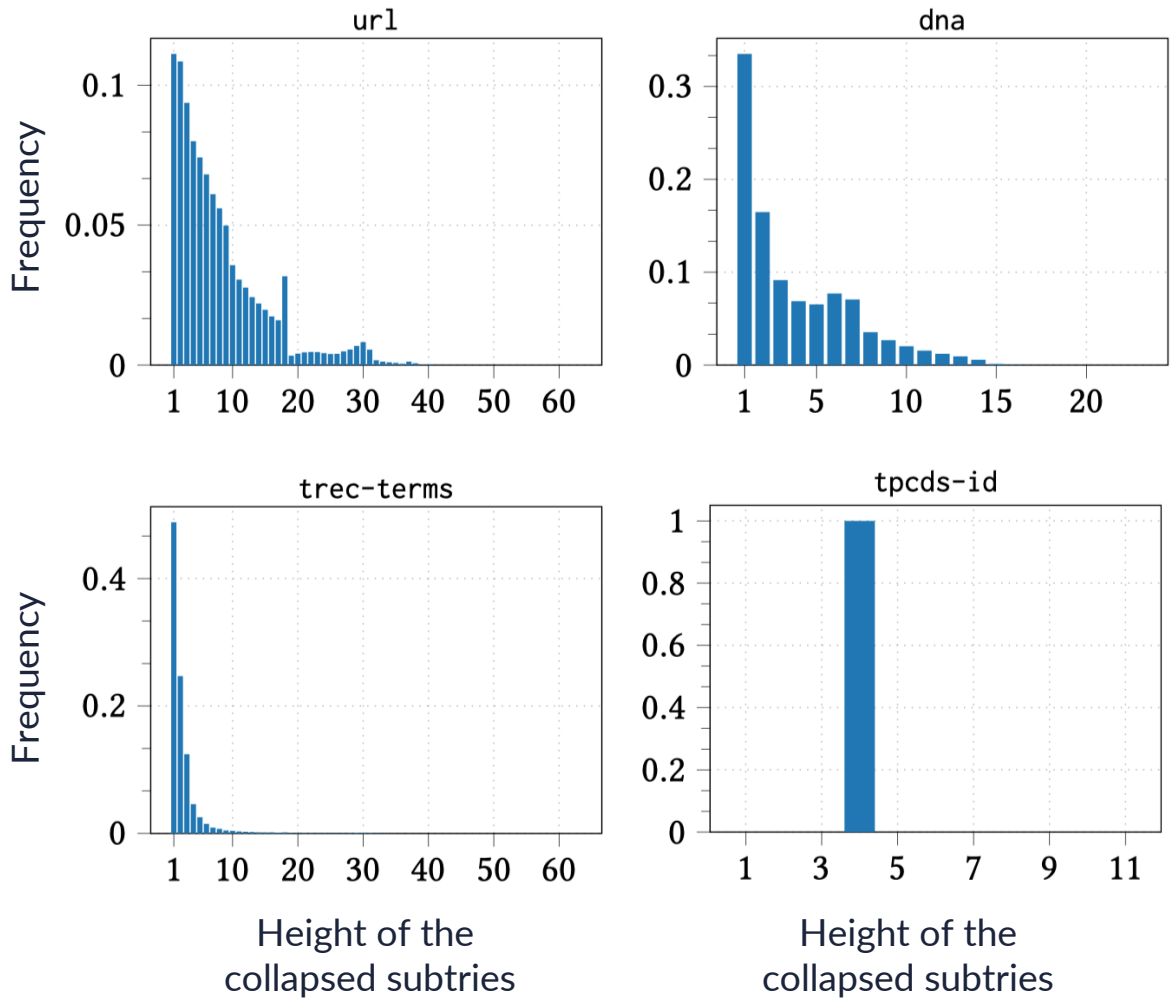
- We aim for the layout that minimise the space (in bytes) of the CoCo-trie
- *Recursive* cost function to evaluate the space of collapsing a single subtrie
- Find the best layout via a bottom-up algorithm

Which subtries to collapse?

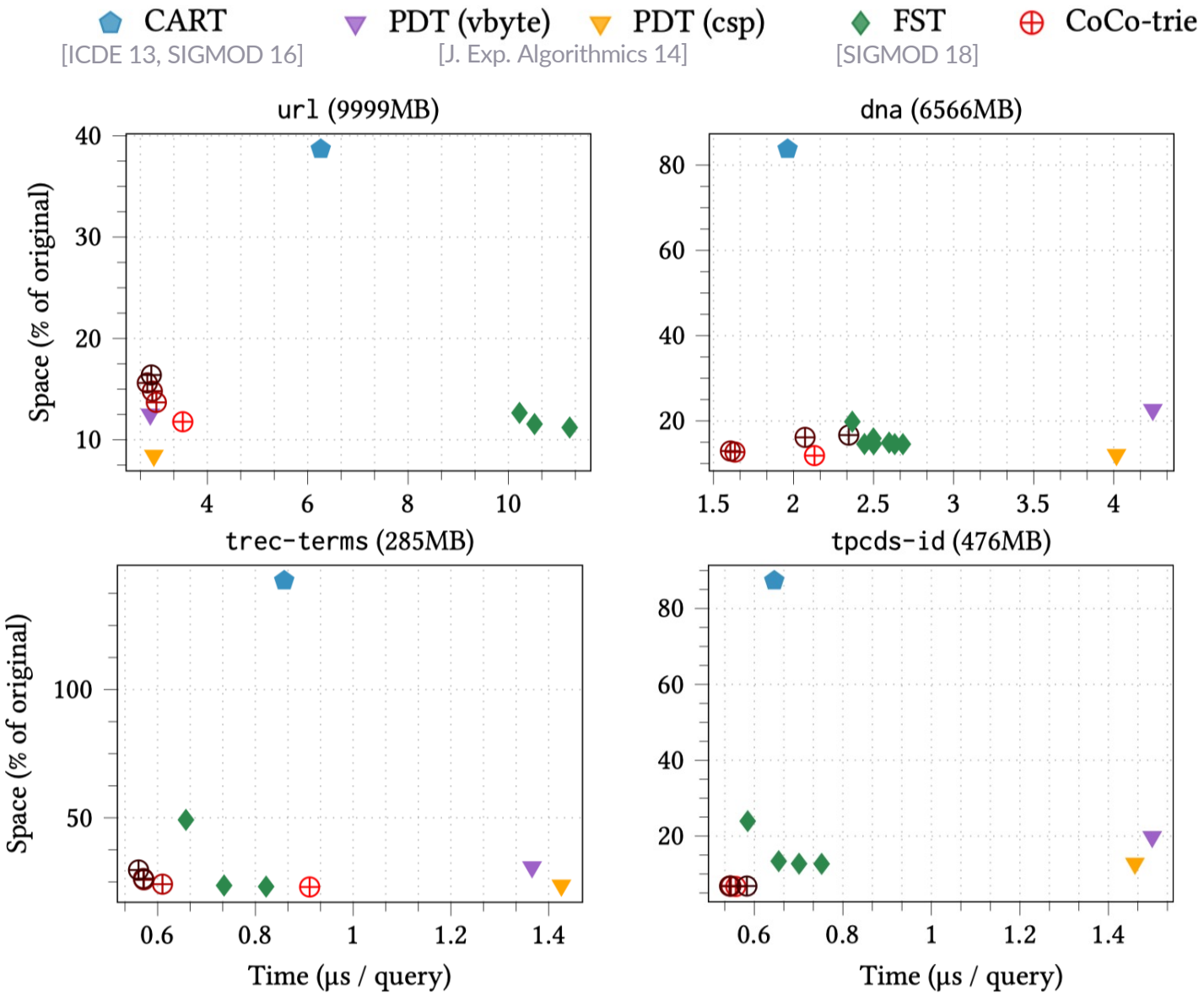


- We aim for the layout that minimise the space (in bytes) of the CoCo-trie
- *Recursive* cost function to evaluate the space of collapsing a single subtrie
- Find the best layout via a bottom-up algorithm

Different datasets, different CoCo-trie layouts



Experiments on space vs query time



CART is not competitive

PDT and FST show inconsistent performance across datasets

CoCo is efficient and flexible

Conclusions

- Advances in compressed indexing schemes
  - for integers: *Block-ε tree*, exploiting data linearity and repetitiveness
  - for strings: *CoCo-trie*, a data-aware compressed trie

Key takeaway:

The performance of classic solutions can be very input-sensitive. New and robust space-time trade-offs by adapting to the data.

Open problems and ongoing work:

- Efficient construction algorithms
- Compress integers with nonlinear models