



Optimally Computing Compressed Indexing Arrays Based on the Compact Directed Acyclic Word Graph

Hiroki Arimura¹

Shunsuke Inenaga²

Yasuaki Kobayashi¹

Yuto Nakashima²

Mizuki Sue¹

1) Graduate School of IST,
Hokkaido University, Japan

2) Department of Informatics,
Kyushu University, Japan

The full paper: <https://arxiv.org/abs/2308.02269>

This slide pdf: <https://ikndeva.github.io> (or arXiv entry's "Code, Data, Media/Paper with Code" section)

This work is partly supported by MEXT Grant-in-Aid for Basic Research A, 2000-2004, Japan



Backgrounds

- Increasing amount and types of **repetitive texts**
 - Markup texts (Wikipedia), Genome sequences
- Development of **compressed index structures** for **repetitive texts** attracts much attention. E.g.,
 - RL-BWT, irreducible PLCP arrays, Lex-parse – size r
 - LZ-parse (LZ76) – size z
 - CDAWG (Compact Directed Word Graphs) – size e

These indices can compress highly-repetitive texts beyond the entropy bounds up to r , z , and e
- Natural questions: What is **the relationships among their sizes?**; what is **the complexities of conversion?**

Backgrounds: Brief History

- We focus on the relationship between three compressed indices.

Suffix
tree
Size n

- BWT is the array of the preceding letters at the starting positions in SA
- r is the number of equal-letter runs

RL-BWT
 r

Irr.
PLCP
 r

- An automata-based index, obtained from the Suffix Tree of T by merging isomorphic subtrees

CDAWG
 e

- LZ-parse is a macro scheme based on the previous factors.
- z is the number of equal-letter runs

LZ-parse
 z

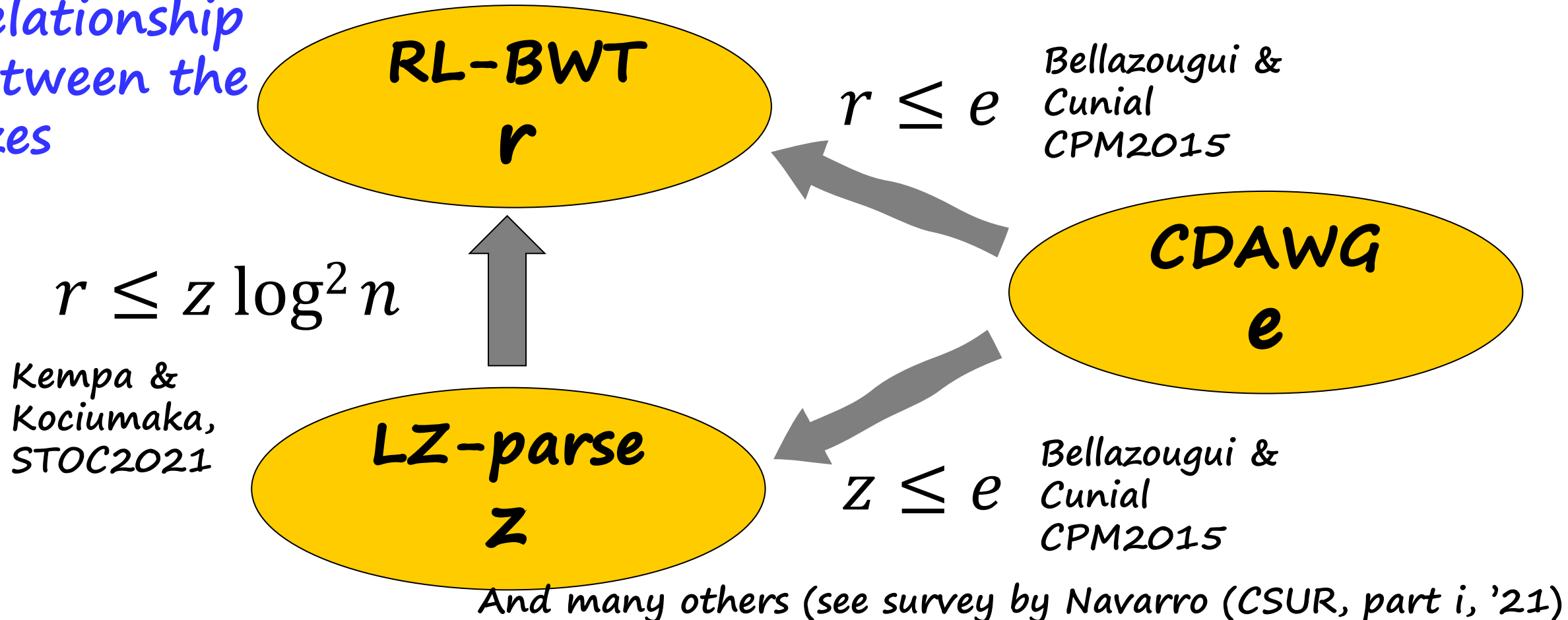
q-Irr.
LPF
 e

- μ : the number of nodes = #maximal extensions
- e is the number of tree- and suffix-edges



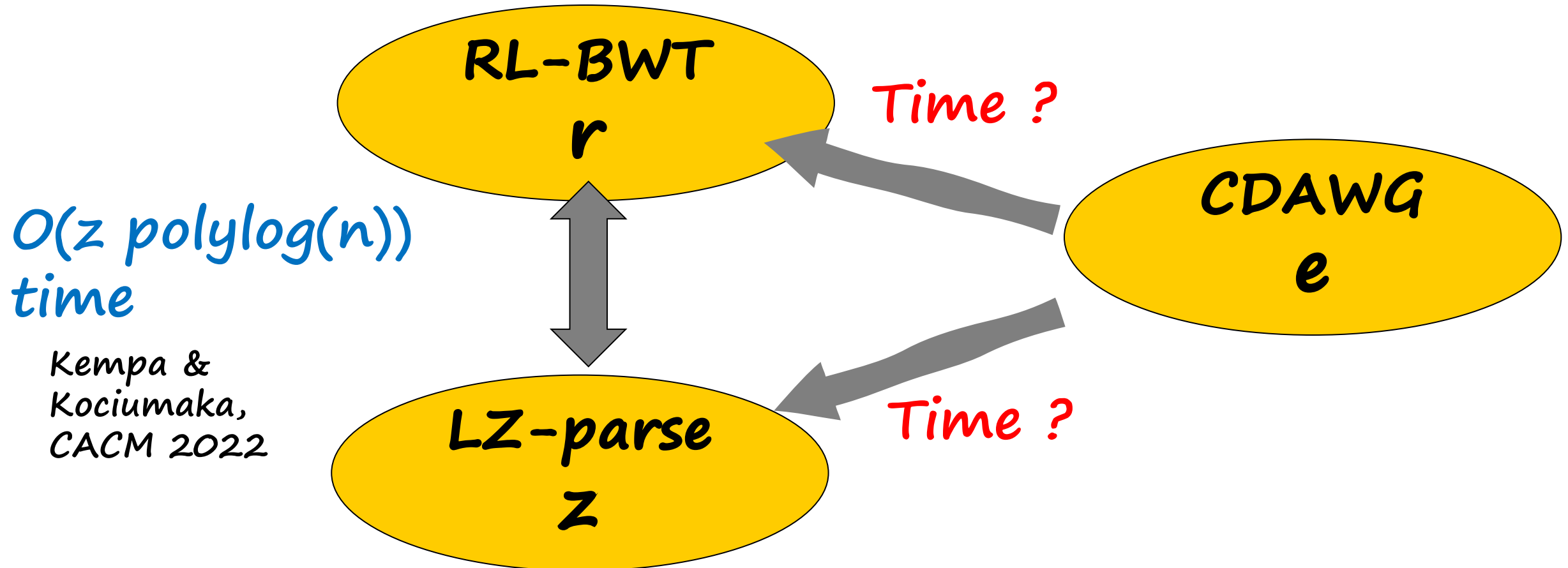
- We focus on the relationship between the indices of the sizes r , z , and e .

Relationship
between the
sizes



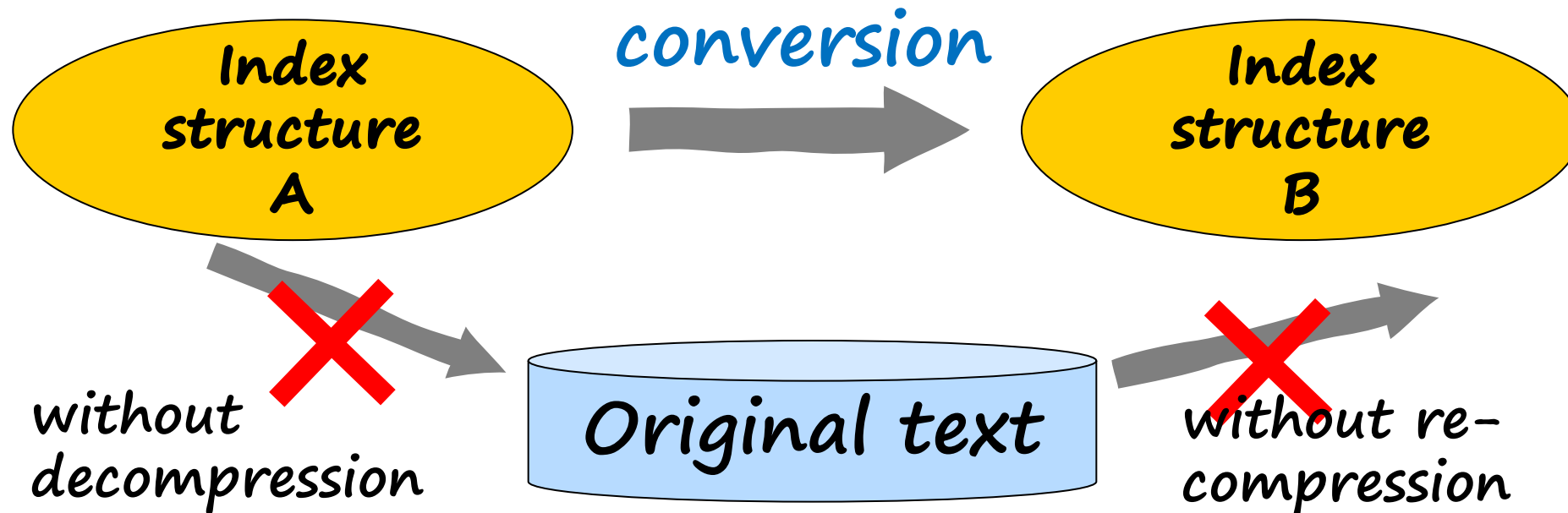
Brief History

- On the other hand, there are not many results on the sub-linear time and space complexities of conversions . . .





- Convert a given compressed index A into another compressed index B without decompression
 - We consider the case that A is the CDAWG of a text T
- Our goal: linear time and space in the combined input and output sizes $|A| + |B|$





Related works

Sublinear time and space conversion between two indices

■ Kempa [SODA'19]

- Converting an RL-BWT-based index into the irreducible PLCP, CSA, and LZ-parse for a text T of length n in $O(n / \log_\sigma n + r \text{ polylog } n)$ time and $O(r)$ space.

■ Kempa & Kociumaka [STOC'21, CACM'22]

- Converting the LZ77-parse of a text T into the RL-BWT for T in $O(z \text{ polylog } n)$ time and space.
- This work solved a long-standing open problem

■ Bannai et al. [CPM'13]

- Converting an SLP of size g into LZ78-parse of size z_{78} in $O(g + z_{78} \log z_{78})$ time and space.
- Combined with Belazzougui & Cunial [CPM'15], we obtain the conversion from the CDAWG for T into LZ78-parse in $O(e + z_{78} \log z_{78})$ time and space.



Thm (4.1, 5.1, 5.2): For any integer alphabet Σ , we can convert **the CDAWG G of size e for a text T** into the following compressed indexing structures for T in $O(e)$ deterministic time and words of space:

- The RL-BWT (run-length BWT) of size r
- The irreducible PLCP (permuted LCP) array of size r
- The quasi-irreducible LPF (longest previous factor) array of size e (def. Sec. 2 of this paper)
- The Lex-parse of size $2r = O(r)$
- The LZ-parse of size z

G is given in either

- the CDAWG of size e with the read only text of length n ,
- the self-index version of CDAWG of size $O(e)$ without a text

Algorithms



Coming back to the relationship
between the sizes ...

Observation: The proof by
Bellazougui & Cunial (2015)
is done by relating “ r ” and
“ z ” to $O(e)$ secondary
incoming/ outgoing edges of
 $CDAWG(T)$

$$r \leq e$$

Bellazougui &
Cunial
CPM2015

CDAWG
 e

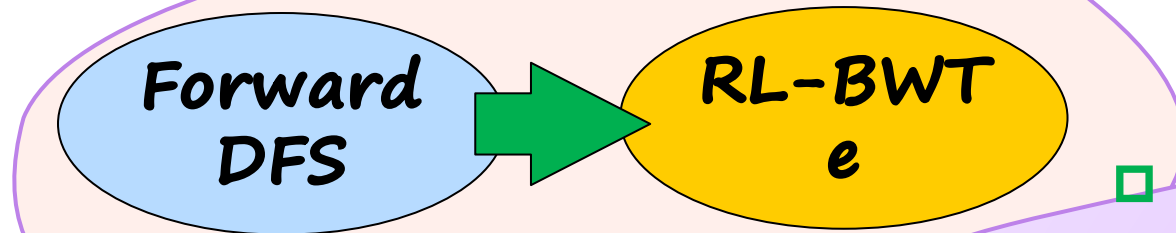
$$z \leq e$$

Bellazougui &
Cunial
CPM2015

LZ-parse
 z

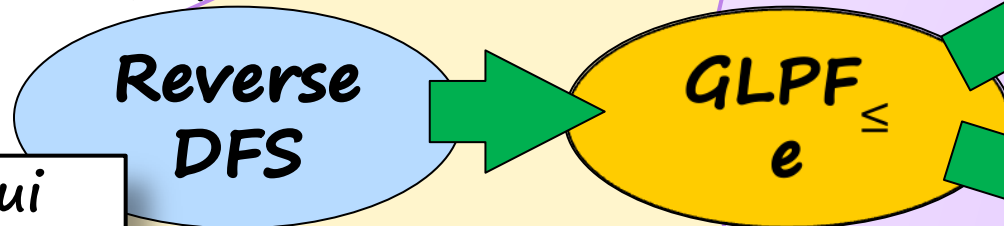
- We use **two orders** of paths
- One for **traversal** of CDAWG
- Order for determining **2ndary edges**

- Order for traversal



Ordered DFS from the source in the lexicographic order

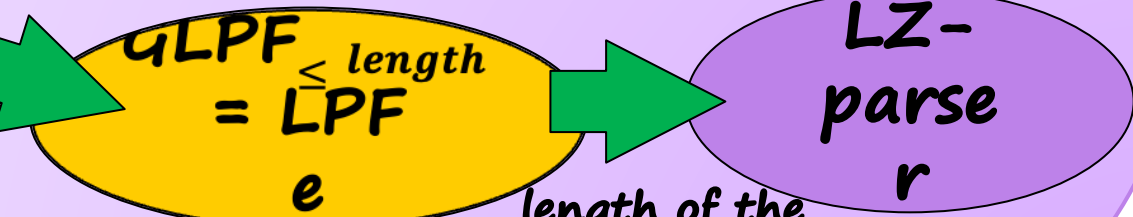
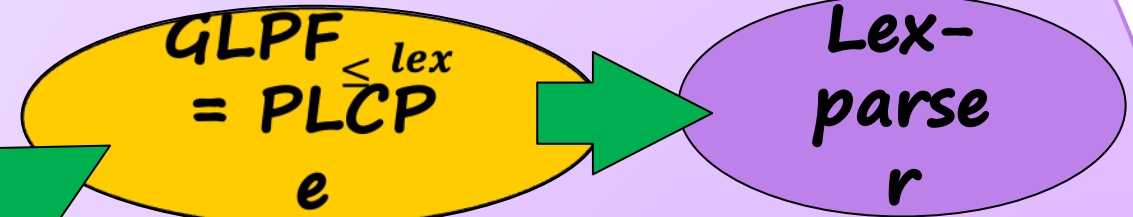
2ndary edge
≈
same-letter run



Ordered DFS from the sink in the text order

Generalized Longest Previous Factor Array [This work]

- Order for 2ndary edges

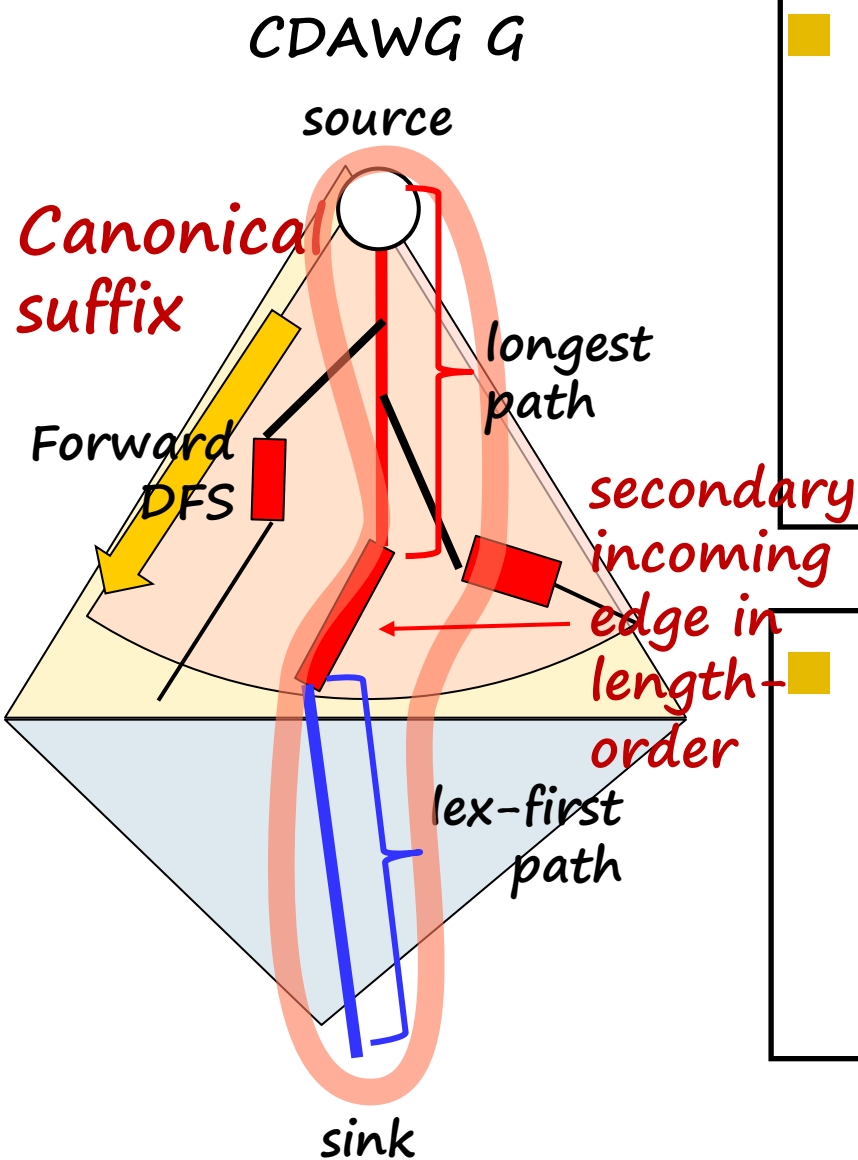


length of the longest upper path ≈ irreducible GLPF-value

Bellazougui & Cunial CPM2015

Navarro, Ochoa, & Prezza (Trans. Inf. Theory, '20).

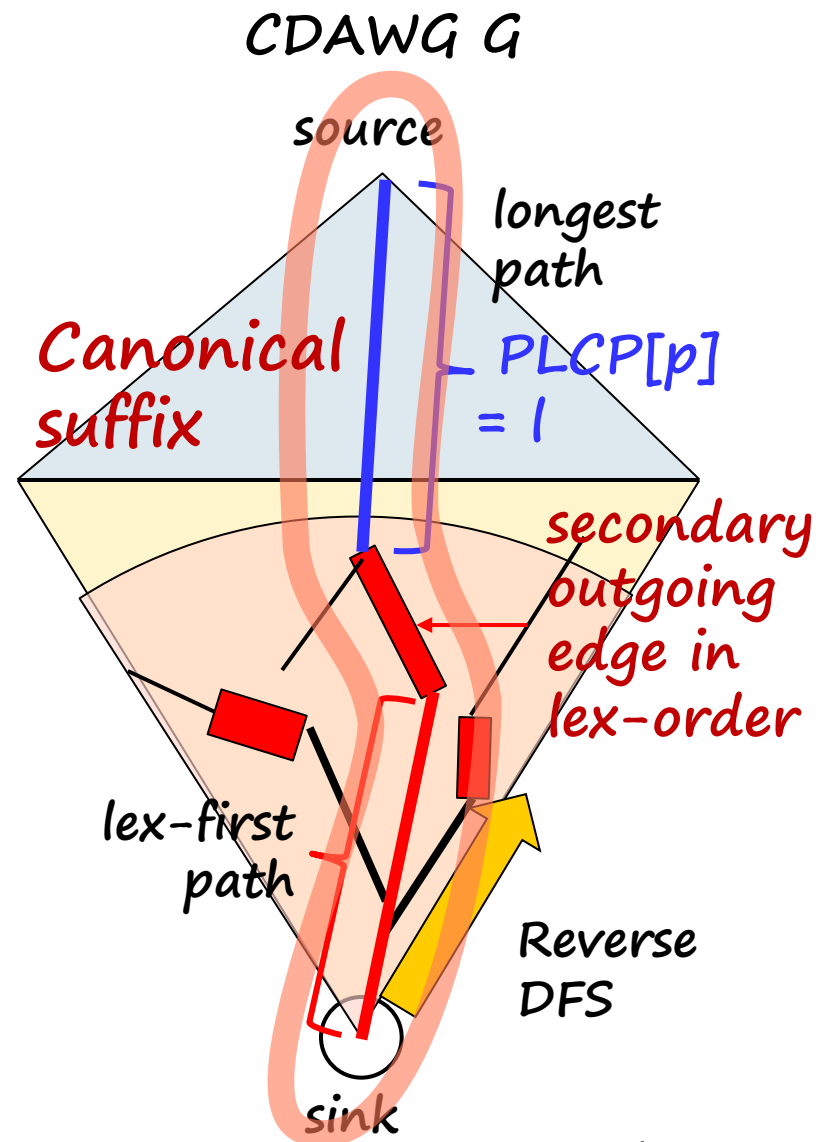
Sec4. Computing RL-BWT in $O(e)$ time&space



■ Observation A1: $O(e)$ secondary incoming edges of $CDAWG(T)$ correspond to subintervals of the same-letter runs of the BWT under the length-order.
(this is because such a search path defines a non-left-maximal factor in T)

■ Observation A2: $O(e)$ incoming edges of $CDAWG(T)$ can be enumerated in the lexicographic order of its "canonical suffix" by the forward DFS from the source.

Sec5. Computing PLCP in $O(e)$ time&space



■ Observation A1: $O(e)$ secondary outgoing edge of $CDAWG(T)$ determines the value $PLCP[p] = l$ by the length l of the longest path from the source to the corresponding branching node under the length-order

■ Observation A2: $O(e)$ secondary outgoing edges can be enumerated in the text order of its “canonical suffix” by the reverse DFS from the sink.

We can extend the above result from PLCP to PLPF by employing the definition of 2ndary outgoing edges in length-order



Conclusions

- Conversion problem from the CDAWG into other compressed indices for highly-repetitive texts
- $O(e)$ time and space conversion from either the CDAWG of a text T or its self-index into the following structures:
 - RL-BWT, (quasi-) irreducible PLCP and LPF arrays, Lex-parse, and LZ-parse for T .
 - Effective version of the result by Belazzougui & Cunial (CPM'15) that $r \leq e$ and $z \leq e$ to actual conversion.
- Techniques:
 - Characterization of the “irreducible values” by secondary edges.
 - Forward/reverse DFS under the lexicographic/text order
- Future Work:
 - Sub-linear time and space conversion from RL-BWT and LZ-parse into CDAWG.

Thank you!

