Grammar Index By Induced Suffix Sorting

Tooru Akagi[1], Dominik Köppl[2],
Yuto Nakashima[1], Shunsuke Inenaga[1],
Hideo Bannai[2], and Masayuki Takeda[1]

1. Kyushu University
2. Tokyo Medical and Dental University

Grammar Compression

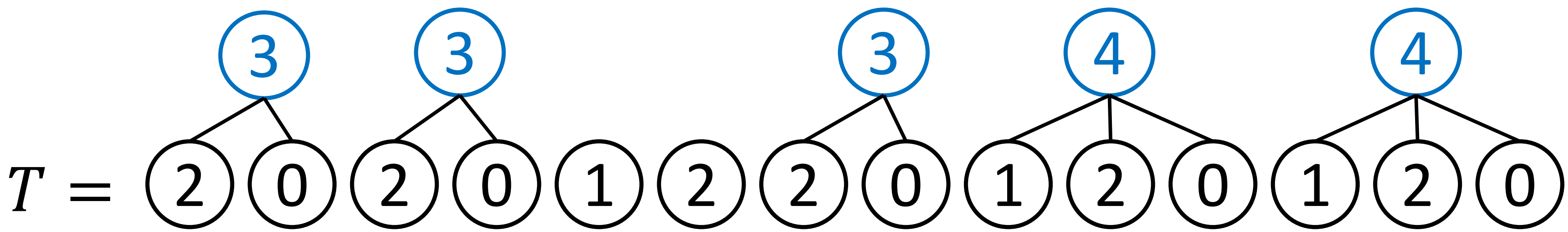
- Given input text T, **Grammar Compression** is a context-free grammar that produces only the original text T.

$T =$ [2 0] [2 0] 1 2 2 0 [1 2 0] [1 2 0]

Grammar Compression

- Given input text T, **Grammar Compression** is a context-free grammar that produces only the original text T.

Non-terminal symbols



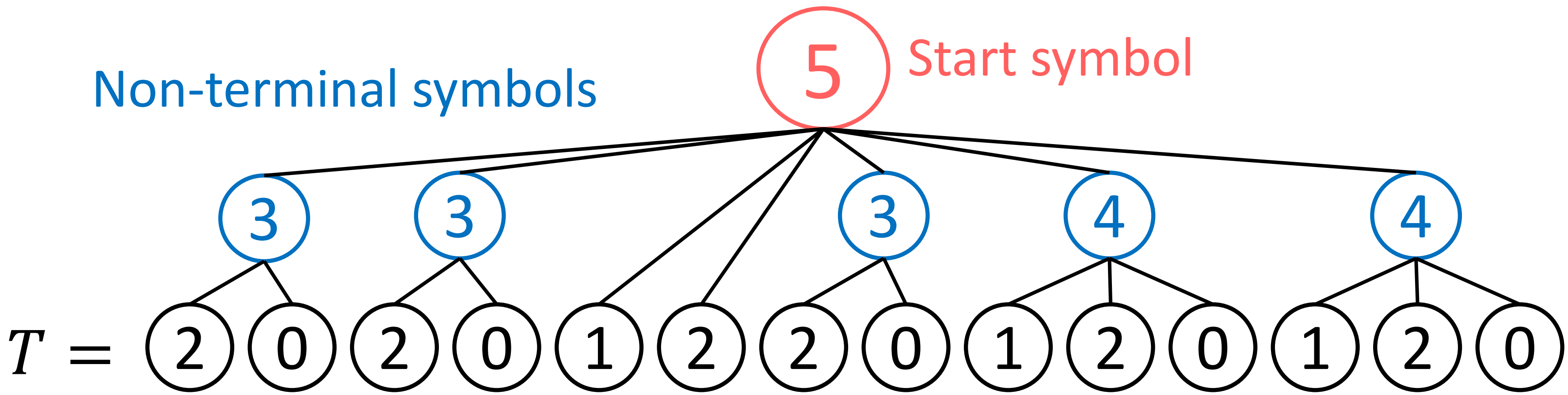
Production rules

$3 \rightarrow 2\ 0$

$4 \rightarrow 1\ 2\ 0$

Grammar Compression

- Given input text T, **Grammar Compression** is a context-free grammar that produces only the original text T.



Production rules

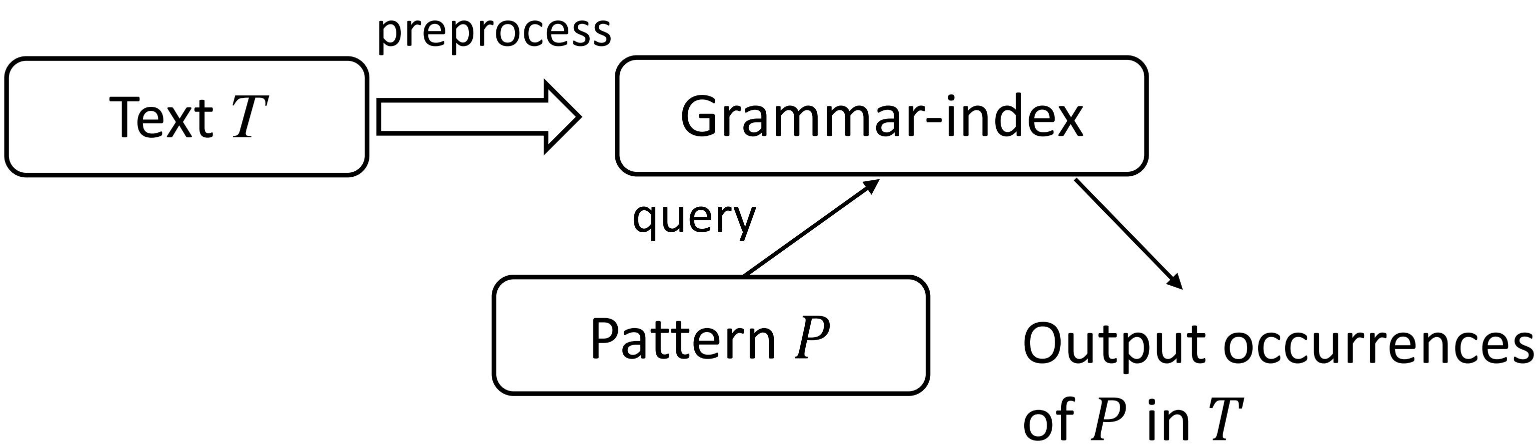
$3 \rightarrow 2\ 0$

$4 \rightarrow 1\ 2\ 0$

$5 \rightarrow 3\ 3\ 1\ 2\ 3\ 4\ 4$

Setting

- Grammar-index

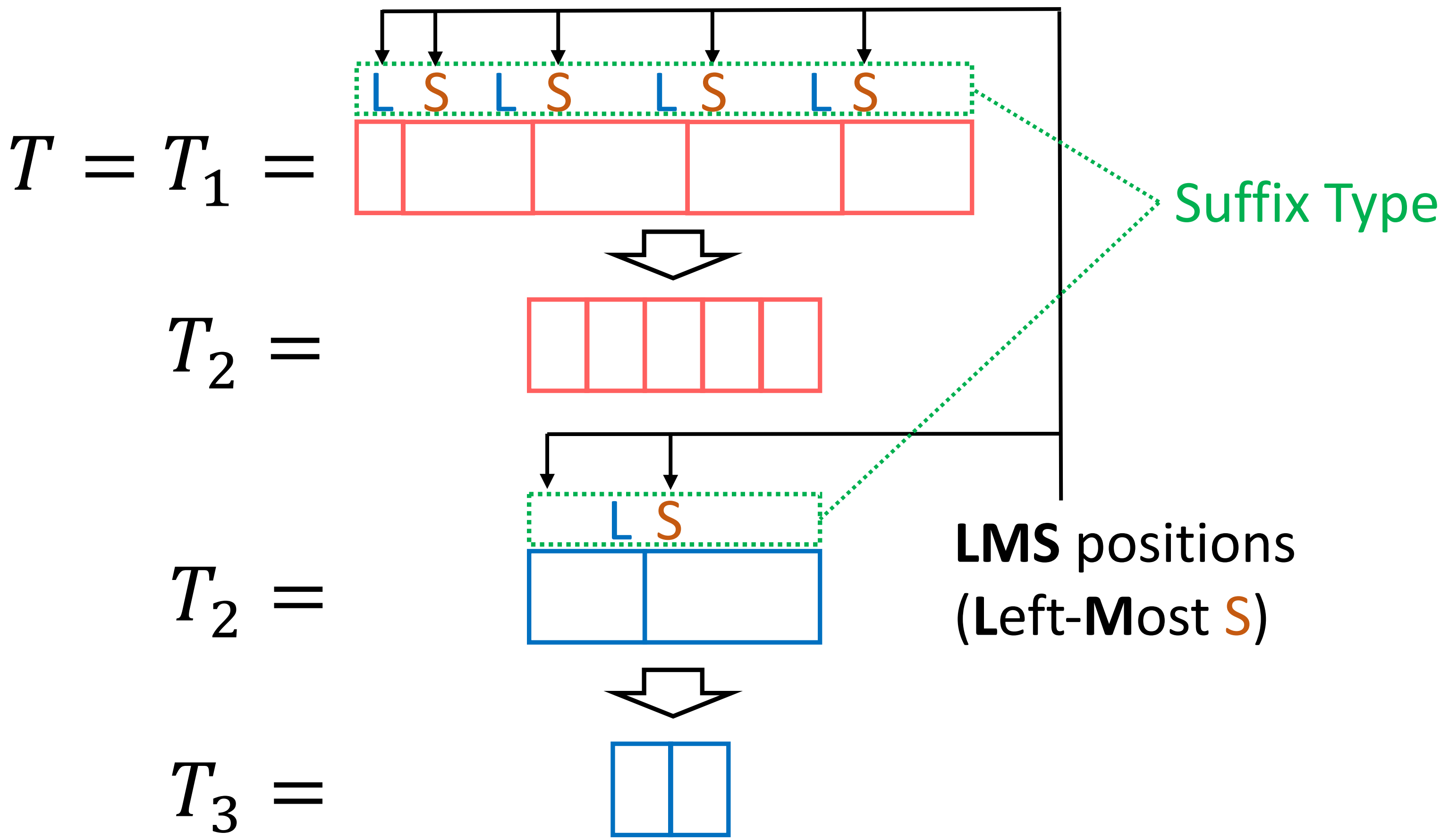


◆We use grammar compression GCIS [Nunes et al. 18], which is based on the suffix sorting algorithm SAIS.

◆We propose a new compressed index called **GCIS-index** and show how to locate all pattern occurrences in T.

GCIS [Nunes et al. 18] (1/3)

GCIS is built by recursively factorizing the text and substituting factors with non-terminals.



Suffix Type

- For every text position i with $1 \leq i \leq |T\#|$:
  1. Type in $T[|T\#|]$ is S.
  2. Type in $T[i]$ is S if $T[i] < T[i+1]$.
  3. Type in $T[i]$ is L if $T[i] > T[i+1]$.
  4. Type in $T[i]$ is the type in $T[i+1]$ if $T[i] = T[i+1]$.

	L	S	S	L	S	L	S	S	L	S	L	S
$T\# =$	3	1	1	3	2	3	1	1	3	2	3	#

Lexicographic order is : $\# < 1 < 2 < \cdots$
(# does not appear in T)

Suffix Type

- For every text position i with $1 \leq i \leq |T\#|$:
  1. Type in $T[|T\#|]$ is S.
  2. Type in $T[i]$ is S if $T[i] < T[i+1]$.
  3. Type in $T[i]$ is L if $T[i] > T[i+1]$.
  4. Type in $T[i]$ is the type in $T[i+1]$ if $T[i] = T[i+1]$.

	L	S	S	L	S	L	S	S	L	S	L	S
$T\# =$	3	1	1	3	2	3	1	1	3	2	3	#

Lexicographic order is : $\# < 1 < 2 < \cdots$
(# does not appear in T)

Suffix Type

- For every text position i with $1 \leq i \leq |T\#|$:
  1. Type in $T[|T\#|]$ is S.
  2. Type in $T[i]$ is S if $T[i] < T[i+1]$.
  3. Type in $T[i]$ is L if $T[i] > T[i+1]$.
  4. Type in $T[i]$ is the type in $T[i+1]$ if $T[i] = T[i+1]$.

	L	S	S	L	S	L	S	S	L	S	L	S
$T\# =$	3	1	1	3	2	3	1	1	3	2	3	#

Lexicographic order is : $\# < 1 < 2 < \cdots$
(# does not appear in T)

Suffix Type

- For every text position i with $1 \leq i \leq |T\#|$:
  1. Type in $T[|T\#|]$ is S.
  2. Type in $T[i]$ is S if $T[i] < T[i+1]$.
  3. Type in $T[i]$ is L if $T[i] > T[i+1]$.
  4. Type in $T[i]$ is the type in $T[i+1]$ if $T[i] = T[i+1]$.

	L	S	S	L	S	L	S	S	L	S	L	S
$T\# =$	3	1	1	3	2	3	1	1	3	2	3	#

Lexicographic order is : $\# < 1 < 2 < \cdots$
(# does not appear in T)

LMS position

- We call position i LMS position if:
  1. $T[i]$ is S but $T[i-1]$ is not S, or
  2. $i = 1$.

	L	S	S	L	S	L	S	S	L	S	L	S
$T\# =$	3	1	1	3	2	3	1	1	3	2	3	#

LMS position

Lexicographic order is : $\# < 1 < 2 < \cdots$
(# does not appear in T)

GCIS [Nunes et al. 18] (2/3)

1. Partition T_i at LMS positions into factors.
2. Sort all factors and replace them in the text with symbols reflecting their lexicographic order.
3. These new symbols induce a new string T_{i+1} we recurse on.

rules

L S S L S L S S L S L S

$$T\# = T_1\# = \boxed{3}\ \boxed{1\ 1\ 3}\ \boxed{2\ 3}\ \boxed{1\ 1\ 3}\ \boxed{2\ 3}\ \#$$

$$T_2\# = 6 \mid 4 \mid 5 \mid 4 \mid 5\ \#$$

$4 \rightarrow 1\,1\,3$
$5 \rightarrow 2\,3$
$6 \rightarrow 3$

GCIS [Nunes et al. 18] (2/3)

- Example for recursion
- When all factors differ, terminate by setting the right-hand side of the start symbol to the remaining string.

L S S L S L S S L S L S

$$T\# = T_1\# = 3\ 1\ 1\ 3\ 2\ 3\ 1\ 1\ 3\ 2\ 3\ \#$$

$$T_2\# = 6\ \ 4\ \ 5\ \ 4\ \ 5\ \#$$

$$T_3\# = 8\ \ 7\ \ 7\ \#$$

rules

$4 \rightarrow 1\ 1\ 3$

$5 \rightarrow 2\ 3$

$6 \rightarrow 3$

$7 \rightarrow 4\ 5$

$8 \rightarrow 6$

$9 \rightarrow 8\ 7\ 7$

Start Symbol

GCIS [Nunes et al. 18] (3/3)

- Example for recursion
- When all factors differ, terminate by setting the right-hand side of the start symbol to the remaining string.

rules

$4 \rightarrow 1\ 1\ 3$

$5 \rightarrow 2\ 3$

$6 \rightarrow 3$

$7 \rightarrow 4\ 5$

$8 \rightarrow 6$

$9 \rightarrow 8\ 7\ 7$

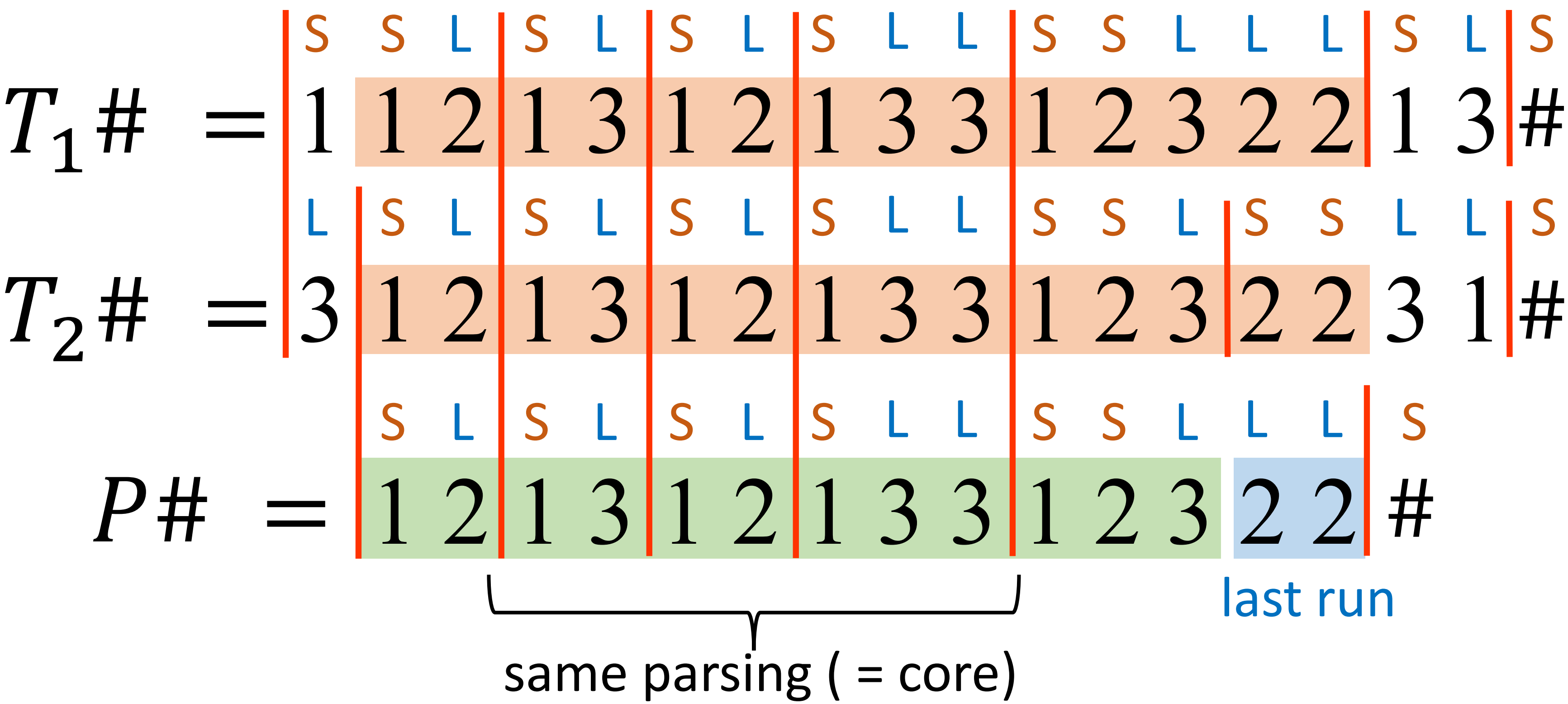
g_{gcis}: The total lengths of all right-hand sides in GCIS-grammar

s_{gcis}: The number of non-terminals in GCIS-grammar

Local consistency in GCIS

We showed the following:

Any occurrence of pattern P in T is parsed in the same way except for the first factor and the last two factors.



Pattern matching with GCIS-index

1. Construct core from P
2. Find all core occurrences in grammar
3. Find primary occurrence candidates
4. Verify primary occurrence candidates
5. Find secondary occurrences

Pattern matching with GCIS-index

1. Construct core from P

   $\Rightarrow$ $O(m \log s_{gcis})$ with GST

2. Find all core occurrences in grammar

   $\Rightarrow$ $O(m \log s_{gcis})$ with GST

3. Find primary occurrence candidates

   $\Rightarrow$ $occ_{gcis} * O(\log n \log s_{gcis})$ with GST

4. Verify primary occurrence candidates

   $\Rightarrow$ $O(m + occ_{gcis} \log m)$ with LCE

5. Find secondary occurrences

   $\Rightarrow$ $O(occ)$ with Jump Pointer

Total Time : $O(m \log s_{gcis} + occ_{gcis} \log n \log s_{gcis} + occ)$

Indexes with Local Consistency Grammars

[Pattern locating time]

GCIS-index (This work)	$O(m \log s_{gcis} + occ_{gcis} \log n \log s_{gcis} + occ)$
ESP-index (Maruyama et al.13)	$O\left((m \log g_{esp} + occ_{esp} \log m \log n) \log^* n\right)$
LMS based self-index (Díaz-Domínguez et al. 21)	$O\left((m \log m + occ) \log g_{gcis}\right)$
Optimal dictionary-compressed index (Christiansen et al. 20)	$O(m + occ)$

occ_{gcis} : # of occurrences of core in GCIS-index

occ_{esp} : # of occurrences core in ESP-index

occ : # of occurrences of patterns

m : pattern length

n : uncompressed text length

g_{esp} : grammar size (ESP-index)

g_{gcis} : grammar size (GCIS)

s_{gcis} : # of non-terminals (GCIS)

Indexes with Local Consistency Grammars

[Index size] (bits)

GCIS-index (This work)	$O(g_{gcis} \log n)$
ESP-index (Maruyama et al.13)	$(1 + \varepsilon) g_{esp} \log g_{esp} + 4 g_{esp} + o(g_{esp})$
LMS-based self-index (Díaz-Domínguez et al. 21)	$g_{gcis} \log n + (2 + \varepsilon) g_{gcis} \log g_{gcis}$
Optimal dictionary-compressed index (Christiansen et al. 20)	$O(\gamma \log(n/\gamma) \log^{(1+\epsilon)} n)$

ϵ : $0 < \epsilon < 1$

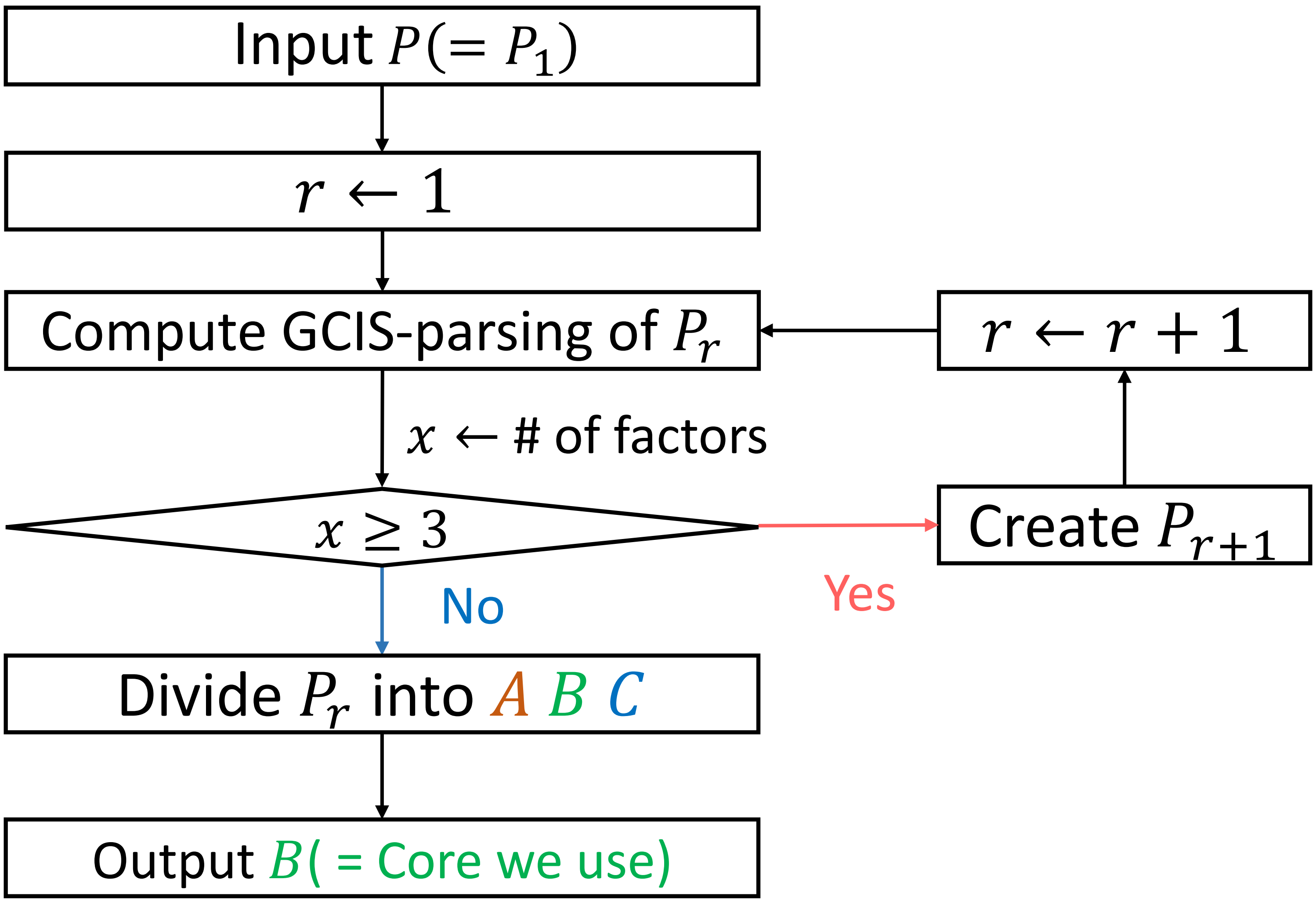
γ : smallest string attractor size

n : uncompressed text length

g_{esp} : grammar size (ESP-index)

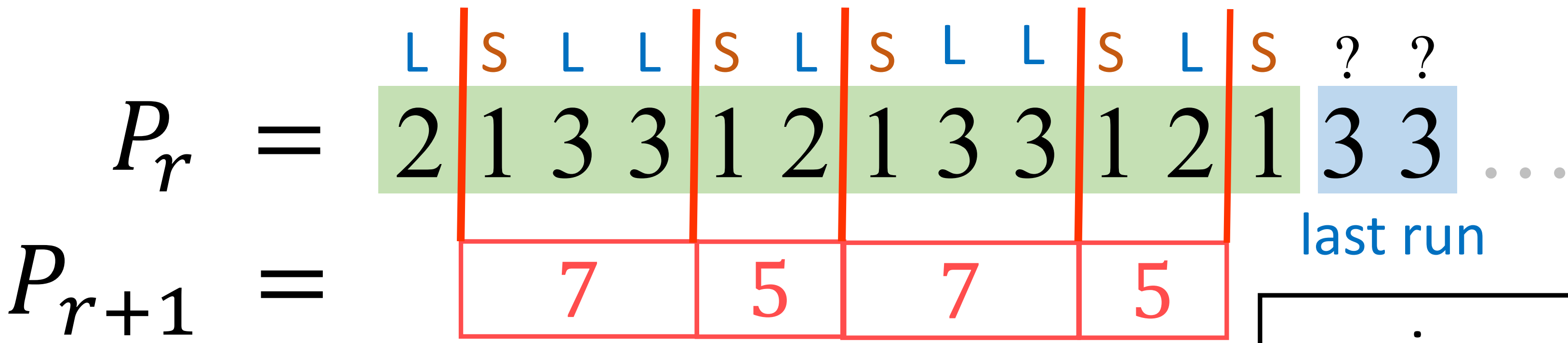
g_{gcis} : grammar size (GCIS)

Computing Pattern Core

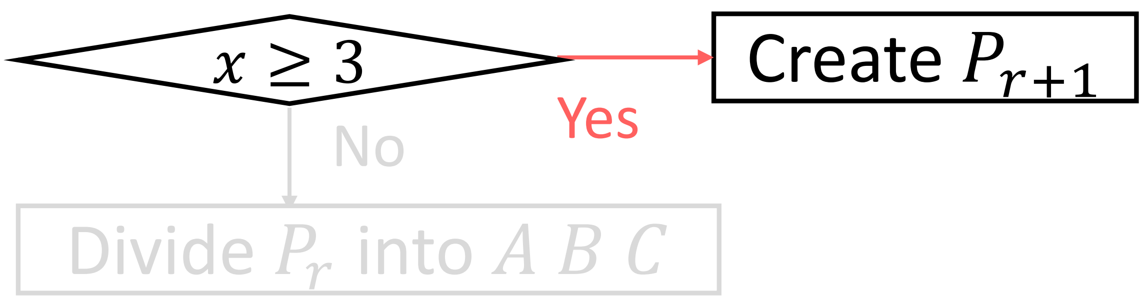


Pattern Core (1/2)

- When we compute the core of pattern P, we use the same non-terminals as in the GCIS for T.

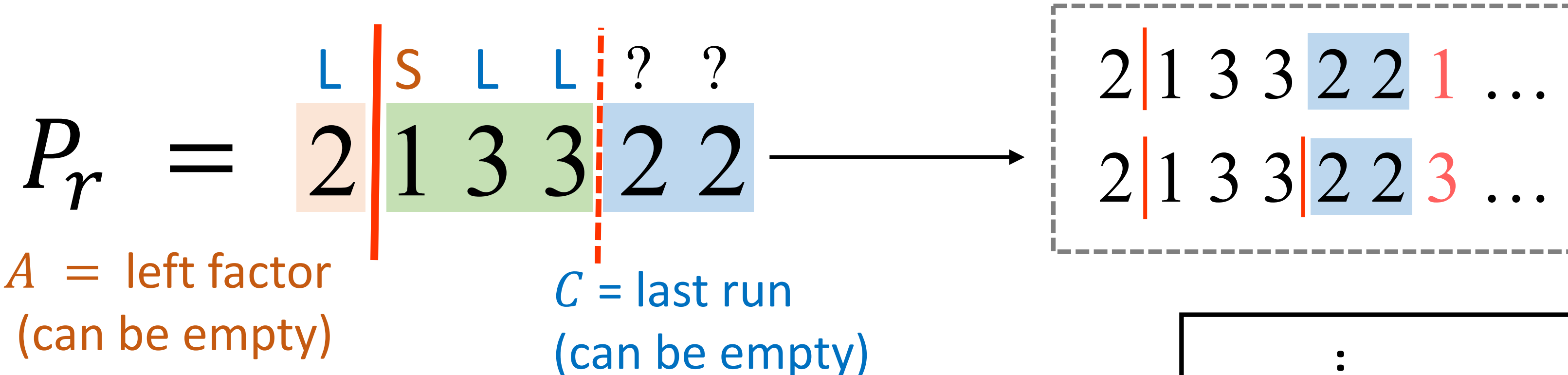


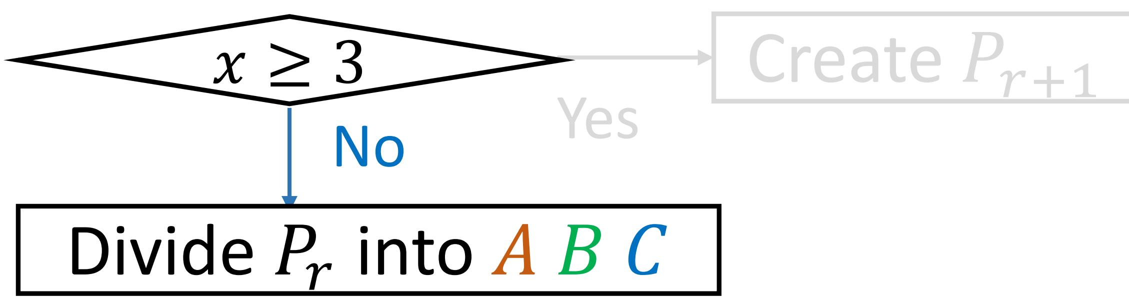
$\vdots$
$5 \rightarrow 1\ 2$
$6 \rightarrow 1\ 2\ 3\ 1$
$7 \rightarrow 1\ 3\ 3$
$8 \rightarrow 1\ 3\ 3\ 2\ 2\ 1$
$\vdots$



Pattern Core (2/2)

- When the number of factors is less than 3, divide the string into 3 parts A B C.





⋮
$5 \rightarrow 1\ 2$
$6 \rightarrow 1\ 2\ 3\ 1$
$7 \rightarrow 1\ 3\ 3$
$8 \rightarrow 1\ 3\ 3\ 2\ 2\ 1$
⋮

Generalized Suffix Tree (GST)

- Suffix Tree of concatenation of the right sides of all production rules.
- For a string B, GST can report all non-terminals having B on their right-hand sides in $O(|B| \log s_{gcis})$ time.

《GCIS-grammar》

⋮

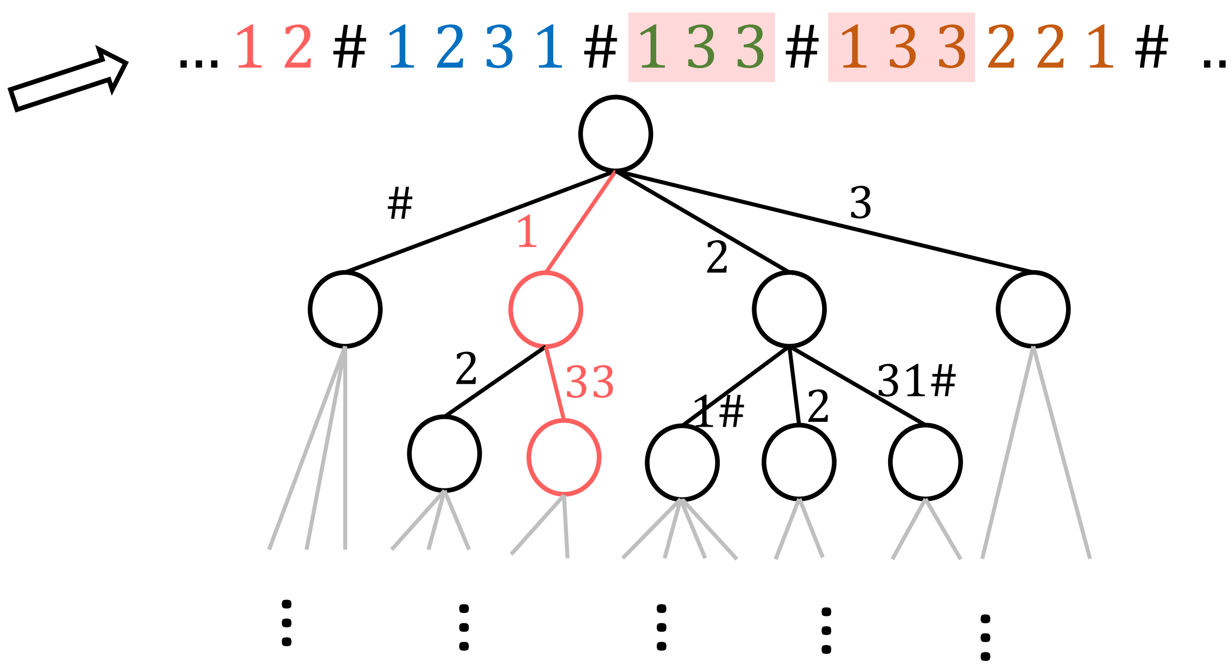
$5 \rightarrow 1\ 2$

$6 \rightarrow 1\ 2\ 3\ 1$

$7 \rightarrow 1\ 3\ 3$

$8 \rightarrow 1\ 3\ 3\ 2\ 2\ 1$

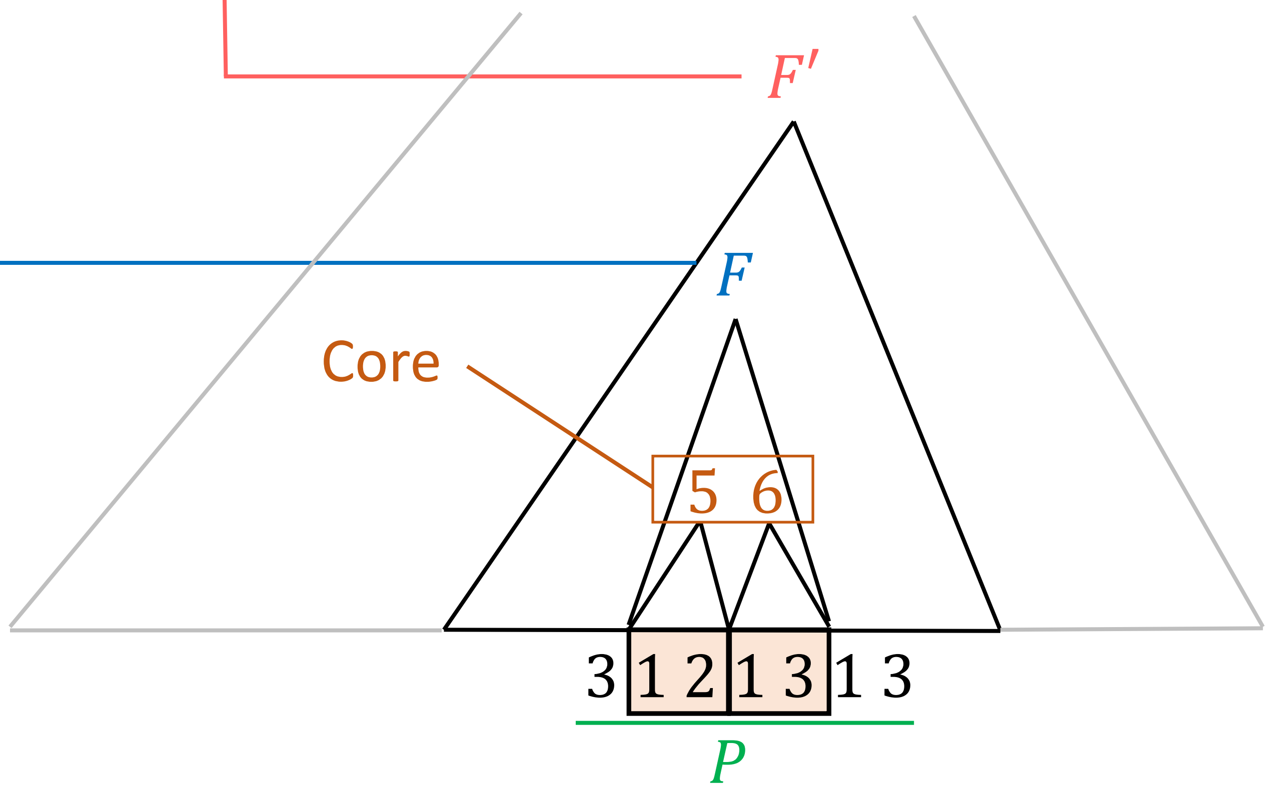
⋮



Primary Occurrence Candidates

- The symbol whose right-hand side contains an occurrence of B might be shorter than the pattern length.

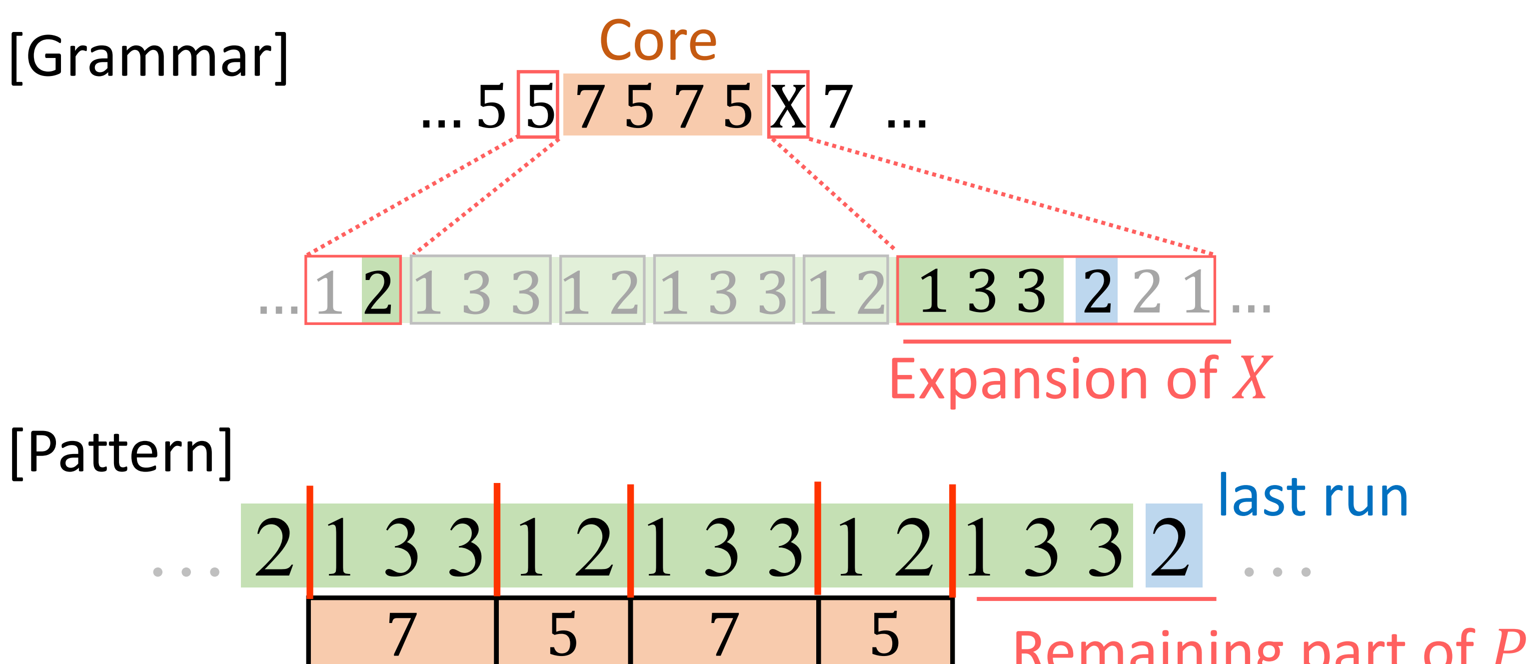
  →climb up the grammar tree to the lowest symbol whose expansion is long enough to contain P



Verify Primary Occurrence Candidates

1. Expand the left & right neighboring symbols of the candidate
2. Check whether they match the remaining part of P
3. Apply recursively

⇨ $O(m\ occ_{gcis})$ time with simple approach



Longest Common Extension

- Longest Common Extension (LCE) query returns longest common prefix of two suffixes.
- Answer in $O(1)$ time by using GST

《GCIS-grammar》

$$\vdots$$
$$X \to 1\ 3\ 3\ 3\ 2\ 1$$
$$Y \to 1\ 3\ 2\ 1$$
$$\vdots$$

$$\text{LCE}\big((X,4),(Y,2)\big) = 3$$

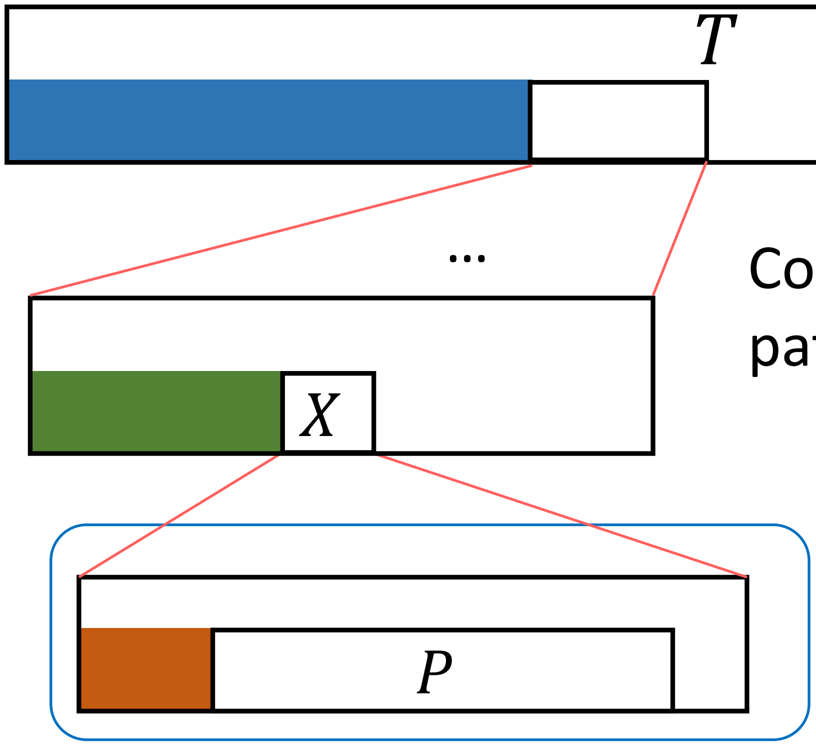
$$X \to 1\ 3\ 3\ \underline{3\ 2\ 1}$$
$$Y \to 1\ \underline{3\ 2\ 1}$$

⇒ LCE speeds up verification of all primary occurrence candidates

$O(m\ occ_{gcis})$ time ⟹ (using LCE) $O(m + occ_{gcis} \log m)$ time

Find Secondary Occurrences

Find secondary occurrences from each primary occurrence
→ We can find all occurrences of P in $O(occ)$ time by application of technique of [Claude and Navarro 12].



Compute starting position of pattern occurrence

X (primary occurrence)

Pattern matching with GCIS-index :Recap

1. Construct core from P

   ⟹ $O(m \log s_{gcis})$ with GST

2. Find all core occurrences in grammar

   ⟹ $O(m \log s_{gcis})$ with GST

3. Find primary occurrence candidates

   ⟹ $occ_{gcis} * O(\log n \log s_{gcis})$ with GST

4. Verify primary occurrence candidates

   ⟹ $O(m + occ_{gcis} \log m)$ with LCE

5. Find secondary occurrences

   ⟹ $O(occ)$ with Jump Pointer

Total Time : $O(m \log s_{gcis} + occ_{gcis} \log n \log s_{gcis} + occ)$

Implementation

- The size of GST is $O(g_{gcis} \log n)$ bits but $O(\cdot)$ hides big constant factor.

→We implement GCIS-nep / GCIS-uni ,which can do pattern matching in GCIS-grammar without GST & LCE queries.

Programming Language : C++

Mac server 2010 with arch linux

Datasets : Pizza Chili and the tudocomp corpus

GCIS-nep

Represents each symbol in 32 bits, and each rule with a 32-bit integer array

GCIS-uni

Encodes all rules by Elias-gamma and Elias-Fano encoding

Index size

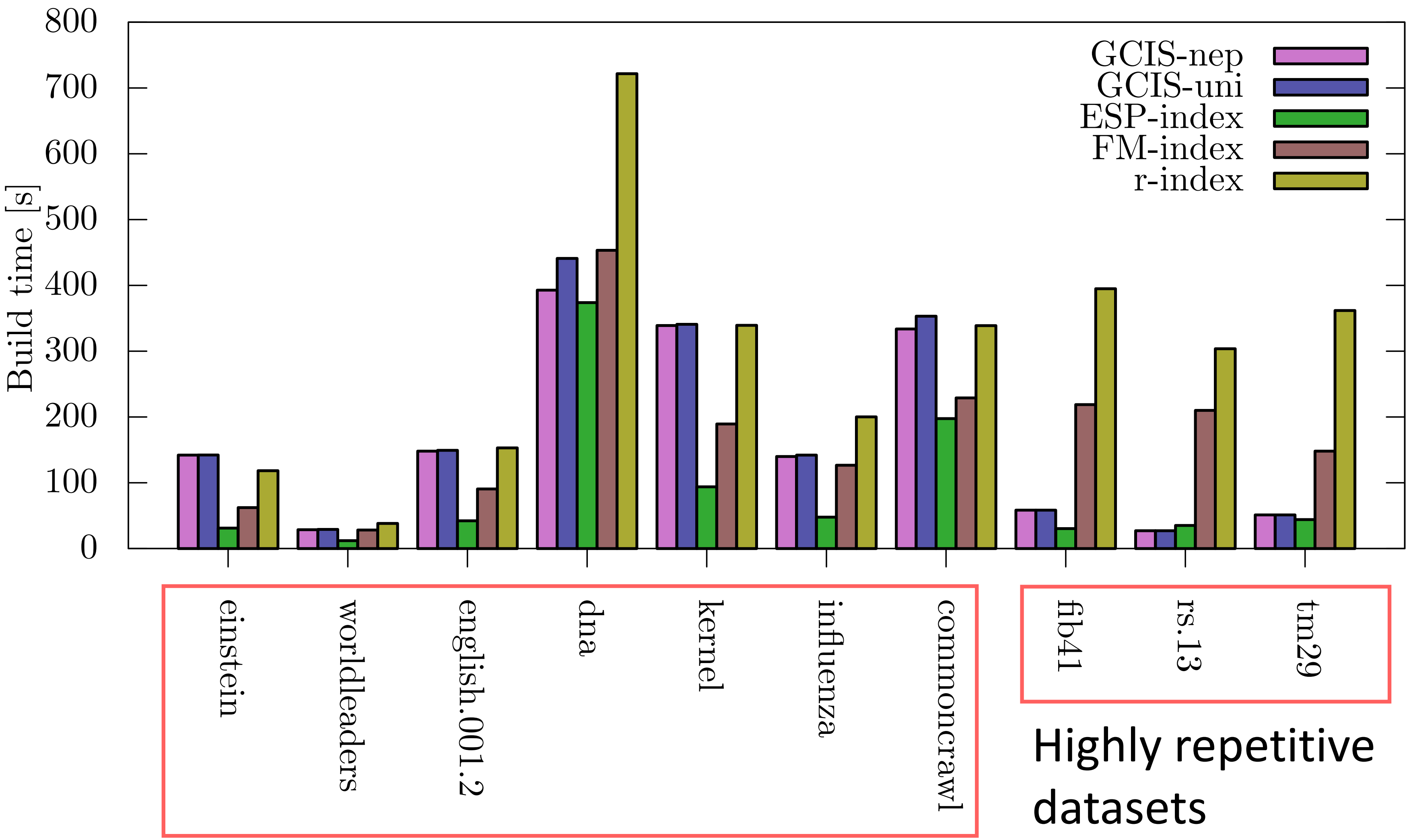
Real and repetitive datasets — Index size (MB)

	Input size	GCIS-nep	GCIS-uni	ESP-index	FM-index	r-index
einstein	92.76	1.13	**0.42**	0.69	40.29	1.14
worldleaders	46.97	5.41	**2.57**	3.61	21.09	5.62
english.001.2	104.86	14.78	**7.48**	10.46	46.98	14.38
dna	403.93	527.55	327.85	297.00	**216.15**	2123.81
kernel	257.96	21.29	**10.46**	12.54	125.08	28.94
influenza	154.81	23.37	**13.87**	15.72	53.06	28.77
commoncrawl	221.18	220.11	138.85	156.00	**122.57**	454.12

Highly repetitive datasets — Index size (KB)

	Input size	GCIS-nep	GCIS-uni	ESP-index	FM-index	r-index
fib41	267914.29	1.37	**0.78**	1.74	71305.79	7.83
rs.13	216747.21	1.73	**0.86**	1.88	57653.91	9.09
tm29	268435.46	2.21	**0.96**	2.19	69347.42	9.17

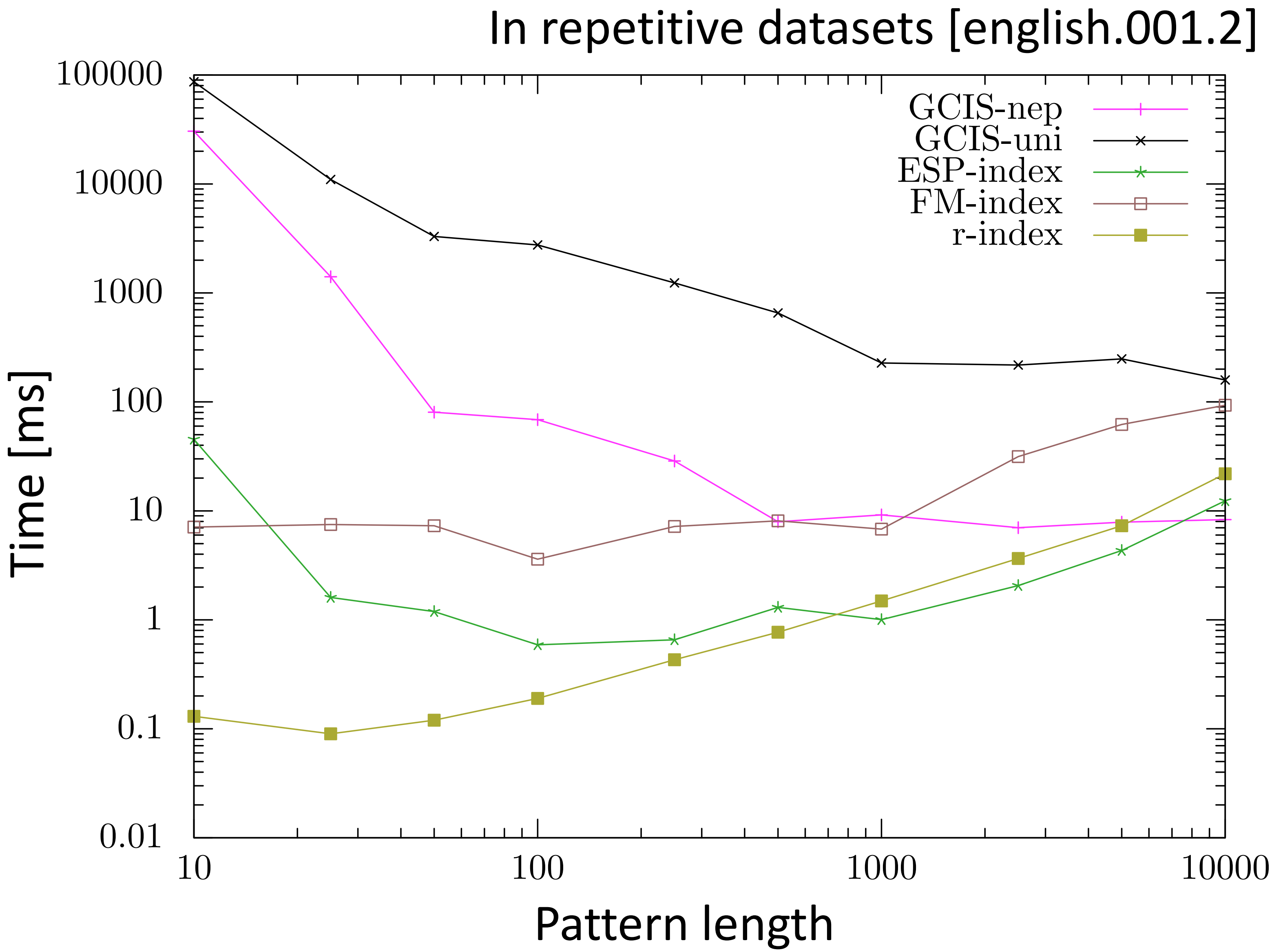
Construction Time



Highly repetitive datasets

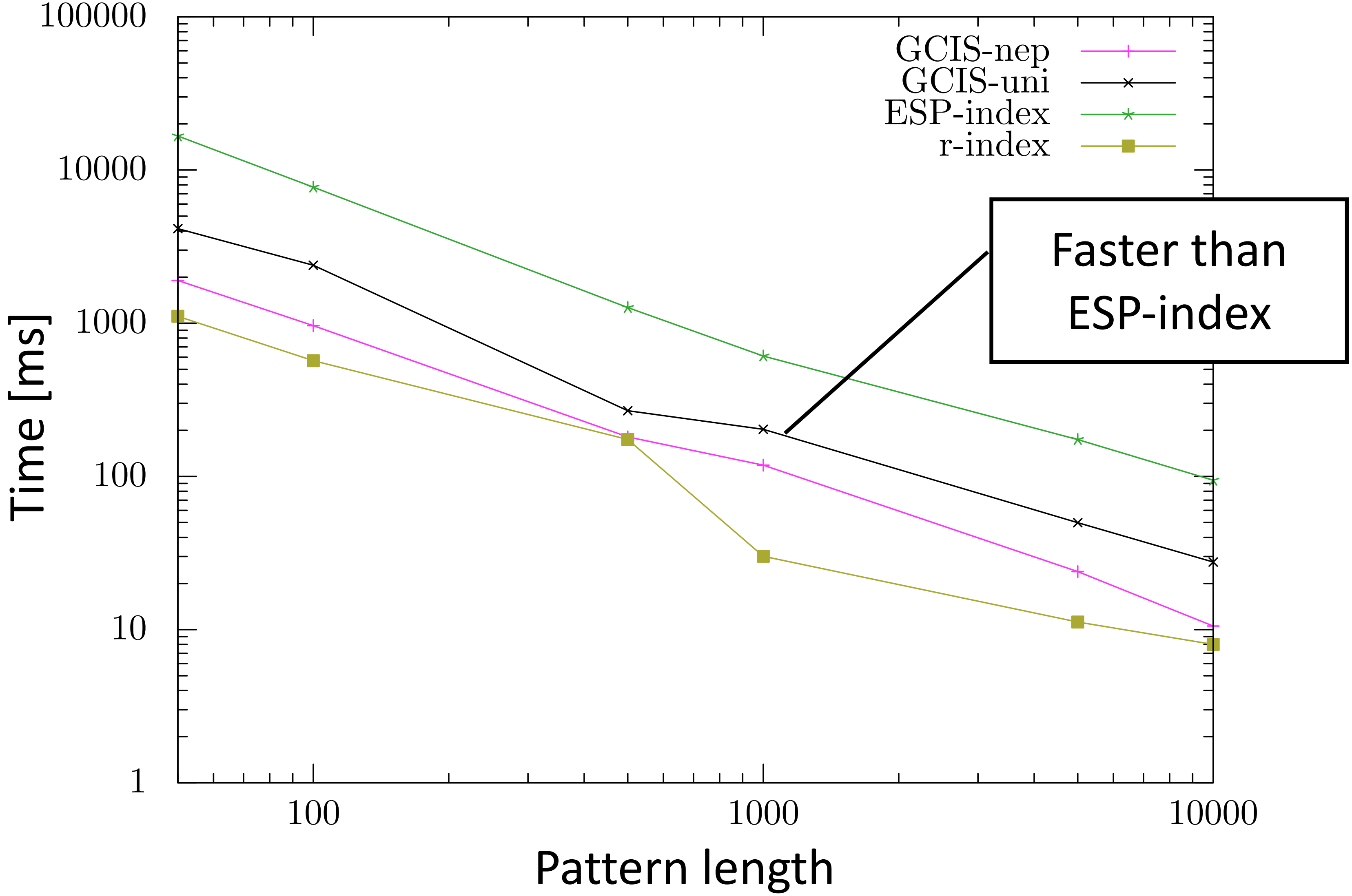
Real and repetitive datasets

Locate Time (1/2)



Locate Time (2/2)

In highly repetitive datasets [fib41]



Summary

- We proposed GCIS-index based on GCIS and showed how to answer locate queries efficiently:

Locate Query: $O(m \log s_{gcis} + occ_{gcis} \log n \log s_{gcis} + occ)$ time

Construction: $O(n)$ time

Space: $O(g_{gcis} \log n)$ bits

- Our implementations GCIS-nep and GCIS-uni are faster than ESP-index when the number of pattern occurrences is large.

  -> Download & try : https://github.com/TooruAkagi/GCIS_Index

Future Work : More experiments, also including

- LMS-based self-index of Díaz-Domínguez et al. (today, 12:30pm)
- the index of Deng et al. (WCTA, 11:30 am)