A Faster Grammar-Based Self-Index

Travis Gagie[1] Paweł Gawrychowski[2] Juha Kärkkäinen[3] Yakov Nekrich[4] Simon Puglisi[5]

Aalto University

Max-Planck-Institute für Informatik

University of Helsinki

University of Bonn

King's College, London

LATA '12

Motivation

We want to store repetitive texts (say, genomic databases) in compressed form, but such that we can search them quickly.
In other words, given a text, build a small structure which allows fast pattern matching.

Pattern matching?

In this talk pattern matching = exact pattern matching, i.e., given $P[1..m]$ we want to find where it occurs **exactly** in text $S[1..n]$.

We might want the first occurrence, or all of them, or just a few...

Such structure is called an index. If it also allows retrieving the original text, it is called a self-index.

Problem, more precisely

We are asked to build an self-index for a string $S[1..n]$ whose LZ77 parse consists of z phrases.

What is LZ77?

Given a text, we split it into z disjoint fragments called phrases. Each fragment is a single letter, or a substring of the already defined prefix.

The number of those phrases is believed to be the **right** measure of how repetitive the text is.

Solution?

Straight-line program, or grammar representation

Simply a context-free grammar with **exactly** one production per nonterminal.

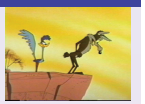
It is known that given a LZ77 parse consisting of z phrases, we can construct such program consisting of just $\mathcal{O}(z \log n)$ words. The program can be assumed to be balanced, meaning that for each production $A \to BC$ we have that $|B| \approx |C|$.

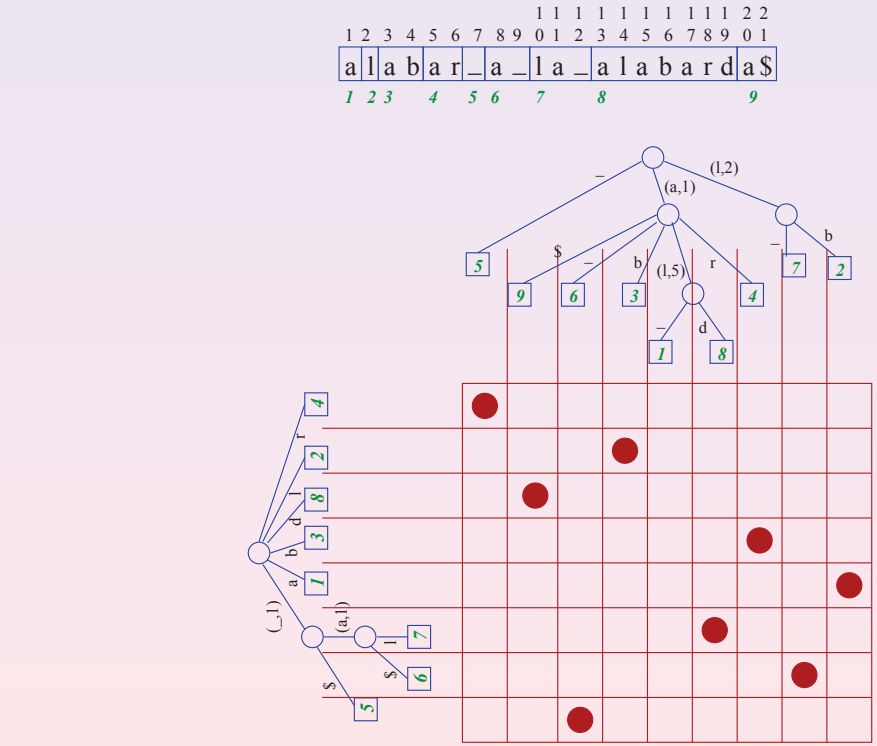
Extracting an arbitrary substring of length ℓ from a balanced SLP takes $\mathcal{O}(\log n + \ell)$ time.

But how do we search?!

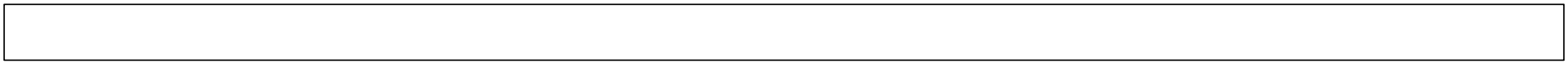
Framework (of Navarro)

A LZ77 Self-Index

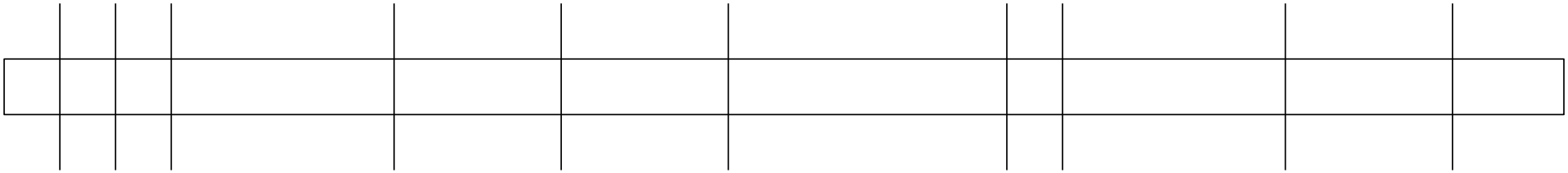




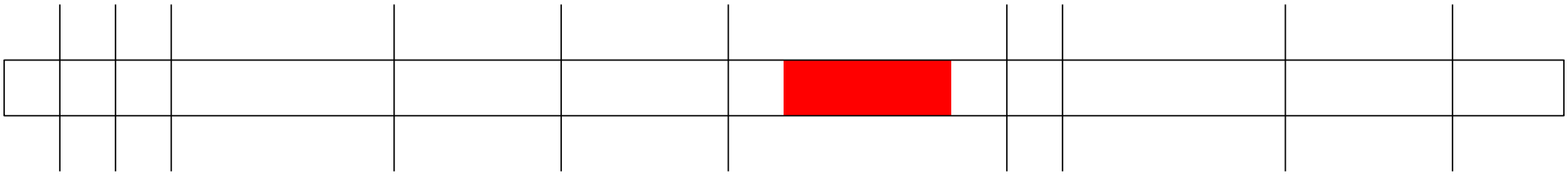
Old Idea



Old Idea



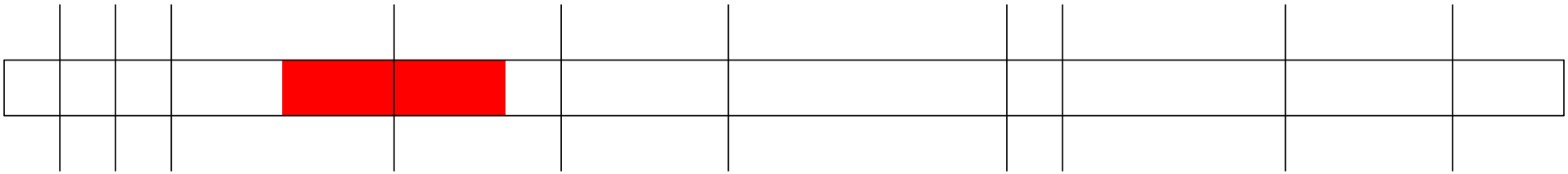
Old Idea



Secondary occurrence

An occurrence is secondary iff it is completely contained in some phrase.

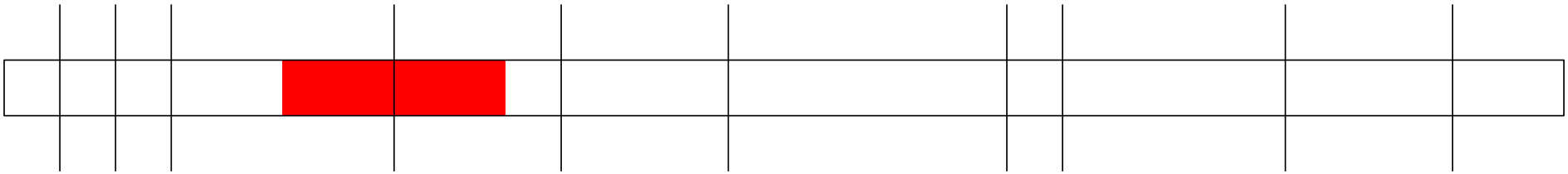
Old Idea



Primary occurrence

An occurrence is primary iff it crosses some boundary.

Old Idea

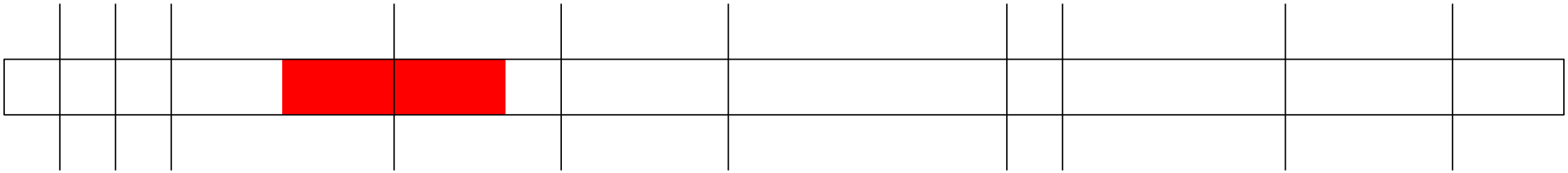


Primary occurrence

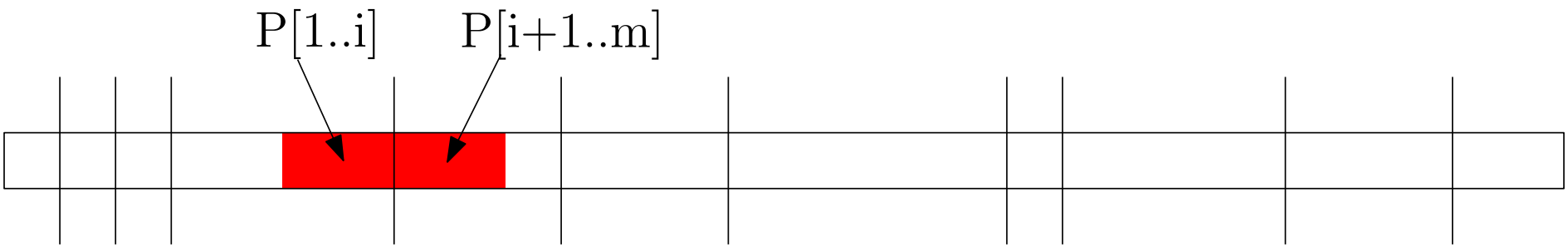
An occurrence is primary iff it crosses some boundary.

Assuming we have all primary occurrences, all secondary occurrences can be found via 2-sided 2D range reporting.

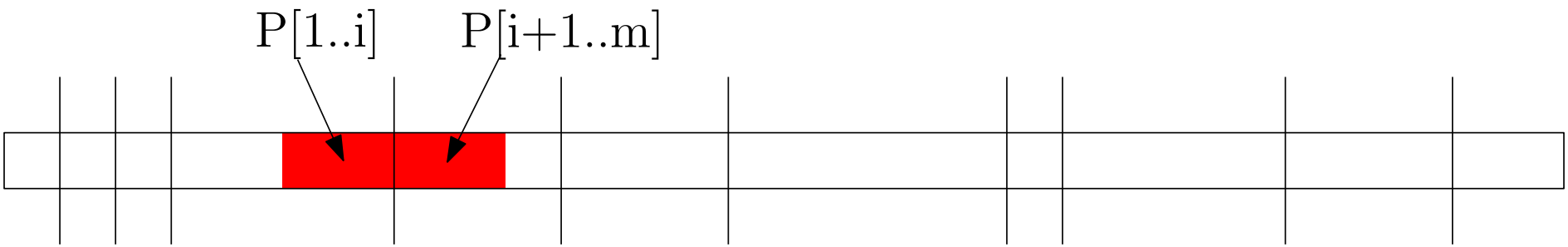
Old Idea, continued



Old Idea, continued



Old Idea, continued



To find all primary occurrences of $P[1..m]$, for each $1 \leq i \leq m$, we

1. search for $P[i+1..m]$ in the Patricia tree of the suffixes starting at phrase boundaries,
2. search for $(P[1..i])^R$ in the Patricia tree of the reversed phrases,
3. check the results via random access,
4. use range reporting to find all boundaries preceded by $P[1..i]$ and followed by $P[i+1..m]$.

New Ideas

We don't use random access during search. We need extract only from the phrase boundaries, so we store bookmarks to them.

We also store a data structure for 1D range reporting (or a bitvector) at each node with depth at most $\log \log z$.

Bounds

Calculation shows that if we choose our data structures carefully . . .

. . . and can extract from bookmarks in $\mathcal{O}(1)$ time per character . . .

. . . then with $\mathcal{O}(z \log \log z)$ extra words we can find all occ occurrences of P in $\mathcal{O}\left(m^2 + occ \log \log n\right)$ time.

Comparison

source	total space (bits)	search time
C & N '09	$\mathcal{O}(\lvert\mathrm{SLP}\rvert) + r \log n$	$\mathcal{O}\big((m^2 + h(m + occ)) \log r\big)$
C & N '11	$(2 + o(1))\lvert\mathrm{CFG}\rvert + R \log n + \epsilon\, r \log r$	$\mathcal{O}\big((m^2/\epsilon) \log R + occ \log r\big)$
K & N '11	$2z \log(n/z) + z \log z + 5z \log \sigma + \mathcal{O}(z) + o(n)$	$\mathcal{O}\big(m^2 d + (m + occ) \log z\big)$
Us	$\lvert\mathrm{BSLP}\rvert + \mathcal{O}(z(\log n + \log z \log\log z))$	$\mathcal{O}\big(m^2 + occ \log\log n\big)$

r: number of rules in the grammar

h: height of the parse tree

R: total length of the right-hand sides

d: depth of nesting in the parse

Bookmarking

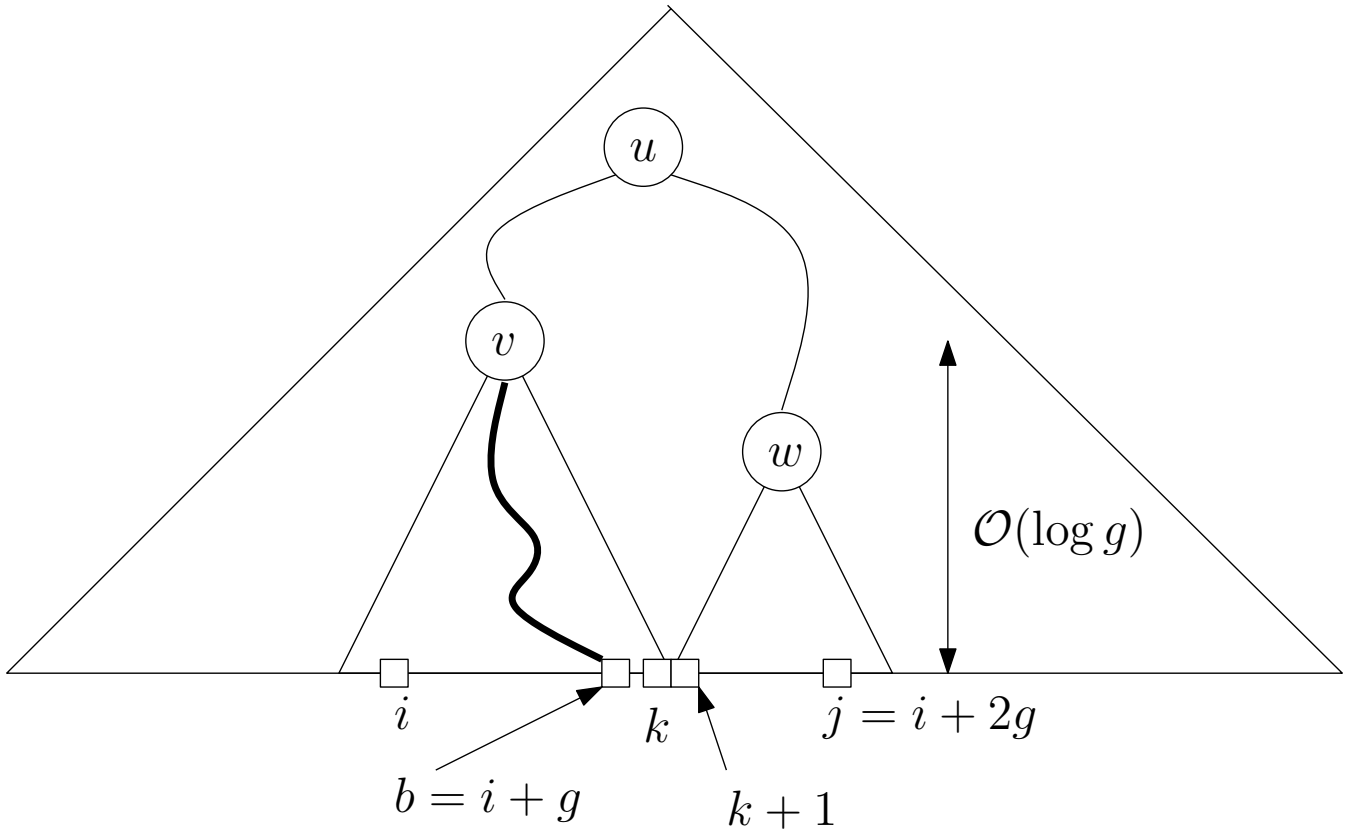
Lemma

Given a balanced SLP for S with r rules and integers b and g, we can store $2 \log r + \mathcal{O}(\log g)$ bits such that later, given $\ell \leq g$, we can extract $S[b - \ell..b + \ell]$ in $\mathcal{O}(\ell + \log g)$ time.

Corollary

We can store $\mathcal{O}(\log^* z)$ words such that, given any ℓ, we can extract $S[b - \ell..b + \ell]$ in $\mathcal{O}(\ell)$ time.

Parse Tree



Space Bounds (in words)

Patricia trees	$\mathcal{O}(z)$
bookmarks	$\mathcal{O}(z \log^* z)$
1D range reporting	$\mathcal{O}(z \log \log z)$
4-sided 2D range reporting	$\mathcal{O}(z \log \log z)$
2-sided 2D range reporting	$\mathcal{O}(z)$
	$\mathcal{O}(z \log \log z)$

Time Bounds

searching in Patricia trees (with perfect hashing if necessary)	$\mathcal{O}(m^2)$
extracting from bookmarks	$\mathcal{O}(m^2)$
1D or 4-sided 2D range reporting	$\mathcal{O}(m^2)$
2-sided 2D range reporting	$\mathcal{O}(occ \log\log n)$
	$\mathcal{O}(m^2 + occ \log\log n)$

Final Result

Theorem

Given a balanced SLP for a string $S[1..n]$ whose LZ77 parse consists of z phrases, we can add $\mathcal{O}(z \log \log z)$ words such that, given a pattern $P[1..m]$, we can find all occ occurrences of P in $\mathcal{O}\left(m^2 + occ \log \log n\right)$ time.

News Flash!!! (not in the paper)

Theorem

Given a balanced SLP with r rules for a string $S[1..n]$ whose LZ77 parse consists of z phrases, we can build an $\mathcal{O}(r + z \log \log z)$-word data structure such that, given a pattern $P[1..m]$, we can find all occ occurrences of P in S in $\mathcal{O}(m \log z + occ \log \log n)$ time.

PAUSE FOR BREATH

Relative Lempel-Ziv*

Kuruppu, Puglisi and Zobel proposed that, to store a genomic database, we

1. build an FM-index for the first genome G (or an artificial reference genome),
2. compress the rest with a version of LZ77 that allows phrases to be copied only from G.

*Do, Jansson, Sadakane and Sung designed an RLZ-index before we did, but didn't publish it.

So what happens in RLZ?

Theorem

We can store the database in

$$\mathcal{O}\big(n(H_k(G)+1) + z(\log n + \log z \log\log z)\big)$$

bits such that we can find all occ occurrences of P in time

$$\mathcal{O}((m+occ)\log^{\epsilon}(n+z)) \ .$$

The $\log^{\epsilon}(n+z)$ factor in the query time comes from accessing the suffix array using Grossi, Gupta and Vitter's CSA.

In real life, it should be $\log n$.

Future work:

1. Is it possible to construct from LZ77 parse a SLP of size smaller than $\mathcal{O}(z \log n)$?
2. Can we achieve $\mathcal{O}(m + occ)$ query time? (maybe with a slightly bigger self-index?)

QUESTIONS?