Bookmarks in Grammar-Compressed Strings

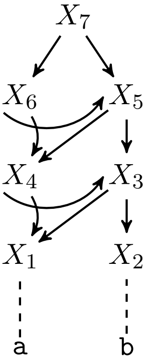
Patrick Hagge Cording

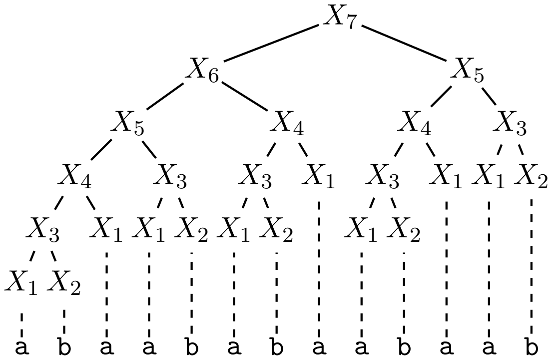
joint work with Pawel Gawrychowski and Oren Weimann

DTU Compute
Department of Applied Mathematics and Computer Science

Grammar compression

$X_1 = \mathtt{a}$
$X_2 = \mathtt{b}$
$X_3 = X_1 X_2$
$X_4 = X_3 X_1$
$X_5 = X_4 X_3$
$X_6 = X_5 X_4$
$X_7 = X_6 X_5$





The grammar has n rules/nodes and generates a string of length N.

Problem and motivation

Problem

- To store a grammar of size n compressing a string of size N, and a set of positions $\{i_1, \ldots, i_b\}$ (*bookmarks*) such that any substring of length l crossing one of the positions can be decompressed in $O(l)$ time.

Motivation

- Compression of several files into one
- Indexes

Results

Random access	Time	Space
Bille et al. [SODA '11]	$O(\log N + l)$	$O(n)$
Belazzougui et al. [ESA '15]	$O(\log_\tau N + l)$	$O(n\tau \log_\tau \frac{N}{n})$
Belazzougui et al. [DCC '14]	$O(l)$	$O(n^{1-\varepsilon} N^\varepsilon)$

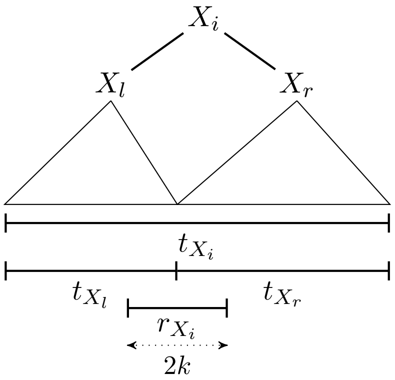
Finger search	Time	Space
Bille et al. [FSTTCS '16]	$O(l \log l)$	$O(n)$

Balancing	Time	Space
Gagie et al. [LATA '12]	$O(l)$	$O(n \log \frac{N}{n} + b \log^* N)$

New result	Time	Space
This	$O(l)$	$O((n+b)\max\{1, \log^* n - \log^*(\frac{n}{b} + \frac{b}{n})\})$

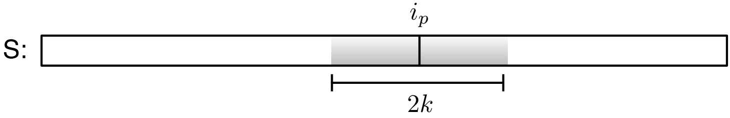
Definition: stabbed substring

For a fixed k, the stabbed substring of a node X_i is the concatenation of the suffix of X_l and prefix of X_r of length k.



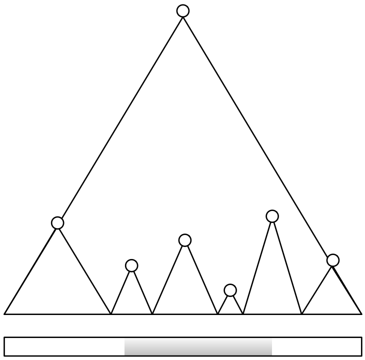
Definition: bookmark substring

For a fixed k, the bookmark substring of a bookmark i_p is the concatenation of the k characters before and after i_p.



Definition: substring cover

For a substring $S[i,j]$, a substring cover for $S[i,j]$ is a set of nodes $X_1, \ldots, X_k$ s.t. $S[i,j]$ is a substring the string $t_{X_1} \ldots t_{X_k}$.



k is the size of the cover.

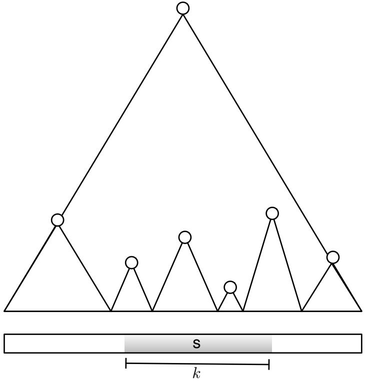
Basic solution

- Store stabbed substring ($k = \log N$) for every node, OR
- Store bookmark substring ($k = \log N$) for every bookmark
- Build $O(\log N)$-time random access data structure of Bille et al.

- Query: if $l > \log N$, use random access. Else read from stored substrings
- Time: $O(l)$
- Space: $O(n + b + \min\{n, b\} \log N)$

SLP block restructuring

Restructure SLP s.t. for any substring s of length k, we can find $O(1)$ nodes covering s



Due to Gawrychowski [SPIRE '12].

Levelled solution: data structure

- Make τ copies of SLP
- Restructure for $k = \log N, \log\log N, \log^{(3)} N, \ldots, \log^{(\tau)} N$
- Build random access data structure for all copies and original SLP
- Use basic solution on level where $k = \log^{(\tau)} N$
- For each level, find and store $O(1)$ nodes covering $2k$ length bookmark substrings

Space

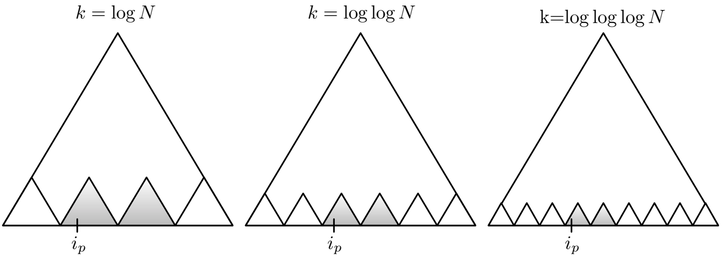
- $O(\tau(n+b) + \min\{n,b\}\log^{(\tau)} N) = O((n+b)\max\{1, \log^* n - \log^*(\frac{n}{b} + \frac{b}{n})\})$

Levelled solution: query

- If $\log^{(k+1)} N < l \leq \log^{(k)} N$ then use random access data structure on level k
- If $l < \log^{(\tau)} N$ then use basic solution

Example

- Decompress $\log\log\log N < l \leq \log\log N$ characters from i_p
- $O(l + \log\log\log N) = O(l)$ time



Summary

Theorem

Given an SLP for $S[1, N]$ with n rules and positions $i_1, \dots, i_b$ in S, we can store S in space $O((n+b)\max\{1, \log^* n - \log^*(\frac{n}{b} + \frac{b}{n})\})$ such that later, given $i \in \{i_1, \dots, i_b\}$ we can extract $S[i, i+l]$ in $O(l)$ time.

Further work

- Remove the $\log^* n$ factor

Thank you.