Compressed Subsequence Matching and Packed Tree Coloring

Philip Bille, Patrick Hagge Cording, and Inge Li Gørtz

DTU Compute, Technical University of Denmark, `phaco@dtu.dk`

CPM, Moscow
June 16, 2014

Subsequence matching

- Given a string S and a pattern P , find all minimal substrings of S where P is a subsequence.
- An occurrence $S[i, j]$ is minimal if P is not a subsequence of $S[i+1, j]$ or $S[i, j-1]$.

Motivation

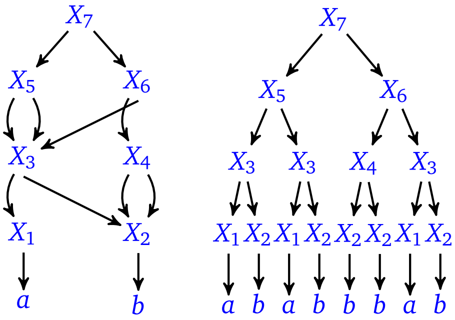
Example: Searching patient logs

- A patient log is a sequence of event the patient has undergone.
- A query could be "give me all patients diagnosed with disease A and given medicine B at least k times before undergoing surgery C (regardless of what happened in between)".
- Pattern is then $P = AB^kC$.

Straight Line Programs

- A grammar in Chomsky normal form that derives one string only.
- Consists of production rules $X_1, \ldots, X_n$ of the form $X_i = X_l X_r$ (nonterminal) or $X_i = \texttt{a}$ (terminal).

Example:



Results

Compressed subsequence matching

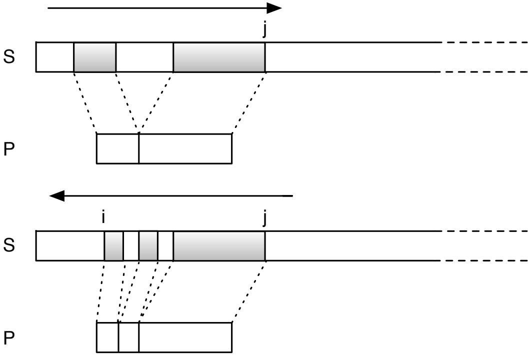
Given a string of size N compressed by an SLP of size n, and a pattern of size m over an alphabet of size σ in the RAM model with word size $w \geq \log N$.

	Time	Space
Cegielski et al.	$O(nm^2 \log m + occ)$	$O(nm^2)$
Tiskin	$O(nm^{1.5} + occ)$	$O(nm)$
Tiskin	$O(nm \log m + occ)$	$O(nm)$
Yamamoto et al.	$O(nm + occ)$	$O(nm)$
This work	$O(n + \frac{n\sigma}{w} + m \log N \log w \cdot \mathrm{occ})$ $O(n + \frac{n\sigma}{w} \log w + m \log N \cdot \mathrm{occ})$	$O(n + \frac{n\sigma}{w})$

Our algorithm

- uses less space and is faster for $occ = o(\frac{n}{\log N})$ (assuming $\sigma \leq m$),
- is output sensitive.

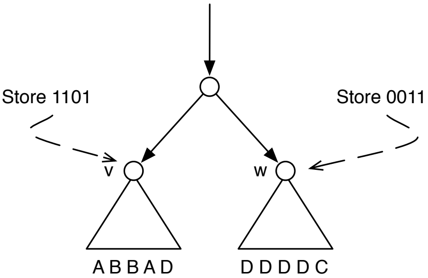
Uncompressed algorithm



- P is a subsequence of $S[1,j]$.
- P is a subsequence of $S[i,j]$ and $S[i,j]$ is a minimal occurrence.
- Algorithm runs in $O(Nm)$ time.

Basic compressed algorithm

- Store a bit-string summary of characters for each node in the SLP.
- Use summaries to find next occurrence of characters in P.



- $O(n + \frac{n\sigma}{w} + mh \cdot occ)$ time and $O(\frac{n\sigma}{w})$ space, where h is the height of the SLP.

New algorithm (1/2)

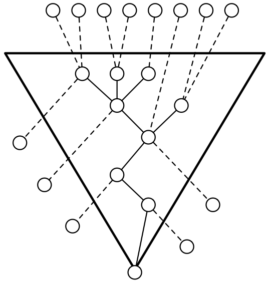
Heavy path decomposition of the SLP

- For production rule $v = uw$, the edge $\{v, u\}$ is heavy if $|S(u)| \geq |S(w)|$, otherwise $\{v, w\}$ is heavy.
- Creates a forest where trees are rooted in the leaves of the SLP – the *heavy forest*.
- At most $\log N$ trees on any path from the root of the SLP to a leaf.

New algorithm (2/2)

Find first occurrence of $P[i]$

- Store summaries of characters to the left and right.
- Query for each tree on path:
  - Find deepest node whose left hanging child (in the SLP) generates $P[i]$.
  - Check if the root of the tree generates $P[i]$.
  - Find the highest node whose right hanging child (in the SLP) generates $P[i]$.



- $O(n + \frac{n\sigma}{w} + p(n) + m \log N \cdot q(n) \cdot occ)$ time and $O(n + \frac{n\sigma}{w} + s(n))$ space.

Tree color problem(s)

Problem and known solutions

Preprocess a colored tree T with t nodes to support first and last colored ancestor queries.

- A first colored ancestor query $\textsc{firstcolor}(v, c)$ is the lowest ancestor of v with color c.
- A last colored ancestor query $\textsc{lastcolor}(u, v, c)$ is the highest node with color c on the path from u to v, where we always assume that u is an ancestor of v.

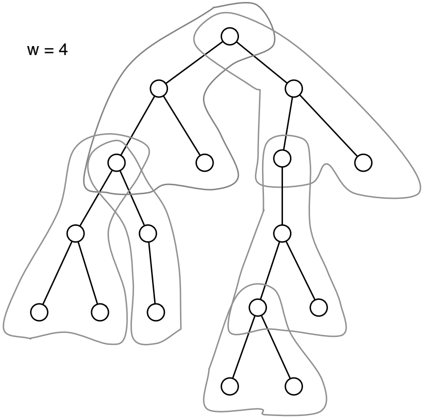
Known solutions:

- $q(t) = O(1)$ query time, $s(t) = O(t\sigma)$ space, $p(t) = O(t\sigma)$ preprocessing time.
- $q(t) = O(\log w)$ query time, $s(t) = O(t + D)$ space, $p(t) = O(t + D)$ preprocessing time.
  - $D = O(t\sigma)$ is the accumulated number of colors used.

Tree color problem(s)

Two level solution

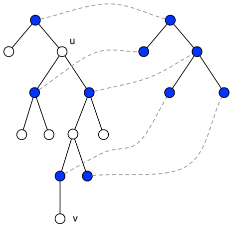
- Convert T to a binary tree T'.
- Partition T' into clusters.
  - Clusters have size $\leq c \cdot w$ for a constant c.
  - Clusters have at most two boundary nodes.
  - Tree comprised of boundary nodes has size $O(\frac{t}{w})$.



Tree color problem(s)

Solution for tree comprised of boundary nodes

- Precompute solutions to FIRSTCOLOR queries.
- For each color c, store the tree consisting only of nodes with color c.
- Build a levelled ancestor data structure for each induced colored subtree.
- A $\text{LASTCOLOR}(u, v, c)$ query:
  - Let $u' = \text{FIRSTCOLOR}(u, c)$ and $v' = \text{FIRSTCOLOR}(v, c)$.
  - $\text{LASTCOLOR}(u, v, c) = \text{LA}(v', \mathit{depth}(u') + 1)$ in the subtree with c-colored nodes.

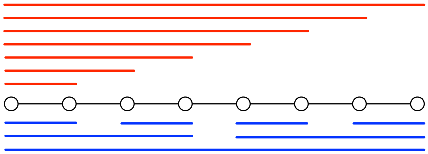


- $O(1)$ query time, $O(\frac{t\sigma}{w})$ space and preprocessing time.

Tree color problem(s)

Solution 1 for clusters

- Make a heavy path decomposition of T.
- For each node, store a bit-string summary of colors of ancestors on heavy path.
- On each heavy path, store a bit-string summary of colors for disjoint subpaths of length $2, 4, \ldots, t$.

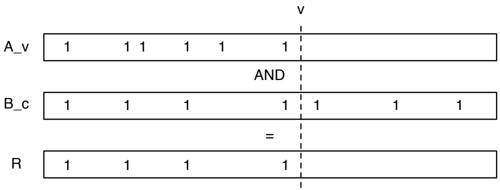


- A FIRSTCOLOR(v, c) query (LASTCOLOR queries are similar):
  - Find deepest heavy path that contains c.
  - Binary search for nearest c-colored node on path.
- $O(\log w)$ query time, $O(\frac{t\sigma}{w})$ space and preprocessing time.

Tree color problem(s)

Solution 2 for clusters

- Assign nodes post-order indices
- Store a bit-string B_c for each color c where $B_c[v] = 1 \iff v$ has color c
- Store a bit-string A_u for each node u where $A_u[v] = 1 \iff v$ is an ancestor of u
- A $\text{FIRSTCOLOR}(v, c)$ query (LASTCOLOR queries are similar):
  - Compute $R = B_c \texttt{ AND } A_v$.
  - Find the index of the least significant set bit in R.



- $O(1)$ query time, $O(\frac{t\sigma}{w})$ space, $O(\frac{t\sigma}{w} \log w)$ preprocessing time.

Summary

The packed tree color problem can be solved using $s(t) = O(t + \frac{t\sigma}{w})$ space,

(i) $q(t) = O(\log w)$ query time and $p(t) = O(t + \frac{t\sigma}{w})$ preprocessing time, or

(ii) $q(t) = O(1)$ query time and $p(t) = O(t + \frac{t\sigma}{w} \log w)$ preprocessing time.

Compressed subsequence matching can be solved using $O(n + s(n))$ words of space and time $O(n + p(n) + m \log N \cdot q(n) \cdot \text{occ})$.

Summary

The packed tree color problem can be solved using $s(t) = O(t + \frac{t\sigma}{w})$ space,

(i) $q(t) = O(\log w)$ query time and $p(t) = O(t + \frac{t\sigma}{w})$ preprocessing time, or

(ii) $q(t) = O(1)$ query time and $p(t) = O(t + \frac{t\sigma}{w} \log w)$ preprocessing time.

Compressed subsequence matching can be solved using $O(n + \frac{n\sigma}{w})$ words of space and time

(i) $O(n + \frac{n\sigma}{w} + m \log N \log w \cdot \text{occ})$, or

(ii) $O(n + \frac{n\sigma}{w} \log w + m \log N \cdot \text{occ})$.