Dynamic Fully-Compressed Suffix Trees

Luís M. S. Russo  Gonzalo Navarro  Arlindo L. Oliveira

INESC-ID/IST
{lsl,aml}@algos.inesc-id.pt

Dept. of Computer Science, University of Chile
gnavarro@dcc.uchile.cl

19th Annual Symposium on Combinatorial Pattern Matching
1 Motivation
   - The Problem We Studied
   - Previous Work and FCST’s
   - Fully-Compressed Suffix Tree Basics

2 Dynamic FCST’s
   - The problem
   - Dynamic CSA’s
   - Updating the sampling

3 Conclusions
   - Summary
Suffix trees are important for several string problems:

- pattern matching
- longest common substring
- super maximal repeats
- bioinformatics applications
- etc
Example (Suffix Tree for $abbbab$)

Luís M. S. Russo, Gonzalo Navarro, Arlindo L. Oliveira

Dynamic Fully-Compressed Suffix Trees
Problem (Suffix Trees need too much space)

*Pointer based representations require $O(n \log n)$ bits.*

This is much larger than the indexed string.
State of the art implementations require $[8, 10] n \log \sigma$ bits.
Sadakane proposed a way to represent compressed suffix trees, in $nH_k + 6n + o(n \log \sigma)$ bits.
A dynamic representation, by Chan et al., requires $nH_k + \Theta(n) + o(n \log \sigma)$ bits and suffers an $O(\log n)$ slowdown.
The Fully-Compressed suffix tree representation requires only \( nH_k + o(n \log \sigma) \) bits.

The representation uses the following scheme:

**Fully-Compressed Suffix Tree**

- Tree Structure
- Compressed Index
- Sampling
  - Nodes represented as intervals
  - LSA
We present dynamic FCST’s that require only $nH_k + o(n \log \sigma)$ bits with a $O(\log n)$ slowdown.
A node represented as an interval of leaves of a suffix tree.

Example

Interval [3, 6] represents node $b$. 

\[
\begin{array}{c}
\text{A:} \\
0 & 1 & 2 & 3 & 4 & 5 & 6 \\
6 & 4 & 0 & 5 & 3 & 2 & 1 \\
\end{array}
\]
Compressed indexes are compressed representations of the leaves of a suffix tree. Their success relies on:

- **Succinct structures**, based on **RANK** and **SELECT**.
- **Data compression**, that represent $T$ in $O(uH_k)$ bits.

### Examples

- FM-index, Compressed Suffix Arrays, LZ-index, etc.

Sadakane used compressed suffix arrays. We need a compressed index that supports $\psi$ and LF. For example the Alphabet-Friendly FM-Index.
Lemma

When $\text{LCA}(v, v') \neq \text{ROOT}$ we have that:

$$\text{SLINK} (\text{LCA}(v, v')) = \text{LCA}(\text{SLINK}(v), \text{SLINK}(v'))$$

This self-similarity explains why we can store only some nodes.
FCST’s use a sampling such that in any sequence

- \( v \)
- \( \text{SLINK}(v) \)
- \( \text{SLINK}(\text{SLINK}(v)) \)
- \( \text{SLINK}(\text{SLINK}(\text{SLINK}(v))) \)
- \( \ldots \)

of size \( \delta \) there is at least one sampled node.
Motivation  Dynamic FCST’s  Conclusions
The Problem  Previous and FCST’s  FCST basics
17 min

Fundamental lemma

**Lemma**

If \( \text{SLINK}^r(\text{LCA}(v, v')) = \text{ROOT} \), and let \( d = \min(\delta, r + 1) \).

Then \( \text{SDep}(\text{LCA}(v, v')) = \max_{0 \leq i < d} \{ i + \text{SDep}(\text{LCSA}(\text{SLINK}^i(v), \text{SLINK}^i(v'))) \} \)

**Proof.**

\[
\begin{align*}
\text{SDep}(\text{LCA}(v, v')) & = i + \text{SDep}(\text{SLINK}^i(\text{LCA}(v, v'))) \\
& = i + \text{SDep}(\text{LCA}(\text{SLINK}^i(v), \text{SLINK}^i(v'))) \\
& \geq i + \text{SDep}(\text{LCSA}(\text{SLINK}^i(v), \text{SLINK}^i(v'))) \\
\end{align*}
\]

The last inequality is an equality for some \( i \leq d \).
Lemma

If $\text{SLINK}^i(LCA(v, v')) = \text{ROOT}$, and let $d = \min(\delta, r + 1)$. Then $\text{SDEP}(LCA(v, v')) = \max_{0 \leq i < d} \{ i + \text{SDEP}(\text{LCSA}(\text{SLINK}^i(v), \text{SLINK}^i(v'))) \}$

Proof.

$\text{SDEP}(LCA(v, v'))$

$= i + \text{SDEP}(\text{SLINK}^i(LCA(v, v')))$

$= i + \text{SDEP}(LCA(\text{SLINK}^i(v), \text{SLINK}^i(v')))$

$\geq i + \text{SDEP}(\text{LCSA}(\text{SLINK}^i(v), \text{SLINK}^i(v')))$

The last inequality is an equality for some $i \leq d$. 

---

Motivation  Dynamic FCST's  Conclusions

The Problem  Previous and FCST's  FCST basics

Luís M. S. Russo, Gonzalo Navarro, Arlindo L. Oliveira  Dynamic Fully-Compressed Suffix Trees
Lemma

If $\text{SLINK}^i(LCA(v, v')) = \text{ROOT}$, and let $d = \min(\delta, r + 1)$. Then $\text{SDEP}(LCA(v, v'))$?

$$\max_{0 \leq i < d}\{i + \text{SDEP}(\text{LCSA}(\text{SLINK}^i(v), \text{SLINK}^i(v'))))\}$$

Proof.

$$\text{SDEP}(LCA(v, v')) = i + \text{SDEP}(\text{SLINK}^i(LCA(v, v'))))$$

$$= i + \text{SDEP}(LCA(\text{SLINK}^i(v), \text{SLINK}^i(v'))))$$

$$\geq i + \text{SDEP}(\text{LCSA}(\text{SLINK}^i(v), \text{SLINK}^i(v'))))$$

The last inequality is an equality for some $i \leq d$. 
Lemma

If $\text{SLINK}^r(LCA(v, v')) = \text{ROOT}$, and let $d = \min(\delta, r + 1)$. Then $\text{SDEP}(LCA(v, v'))$?

$$\max_{0 \leq i < d} \{ i + \text{SDEP}(\text{LCSA} (\text{SLINK}^i(v), \text{SLINK}^i(v'))) \}$$

Proof.

$$\text{SDEP}(LCA(v, v'))$$

$$= i + \text{SDEP}(\text{SLINK}^i(LCA(v, v')))$$

$$= i + \text{SDEP}(LCA(\text{SLINK}^i(v), \text{SLINK}^i(v')))$$

$$\geq i + \text{SDEP}(\text{LCSA} (\text{SLINK}^i(v), \text{SLINK}^i(v')))$$

The last inequality is an equality for some $i \leq d$. 
**Lemma**

If $\text{SLINK}^r(LCA(v, v')) = \text{ROOT}$, and let $d = \min(\delta, r + 1)$. Then $\text{SDEP}(LCA(v, v')) \geq \max_{0 \leq i < d} \{i + \text{SDEP}(\text{LCSA}(\text{SLINK}^i(v), \text{SLINK}^i(v')))\}$

**Proof.**

\[
\text{SDEP}(LCA(v, v')) = i + \text{SDEP}(\text{SLINK}^i(LCA(v, v'))) = i + \text{SDEP}(\text{LCA}(\text{SLINK}^i(v), \text{SLINK}^i(v'))) \geq i + \text{SDEP}(\text{LCSA}(\text{SLINK}^i(v), \text{SLINK}^i(v')))
\]

The last inequality is an equality for some $i \leq d$. 
Lemma

If \( \text{SLINK}^r(LCA(v, v')) = \text{ROOT} \), and let \( d = \min(\delta, r + 1) \). Then \( \text{SDEP}(LCA(v, v')) = \max_{0 \leq i < d}\{i + \text{SDEP}(\text{LCSA}(\text{SLINK}^i(v), \text{SLINK}^i(v'))))\} \)

Proof.

\[
\begin{align*}
\text{SDEP}(LCA(v, v')) &= i + \text{SDEP}(\text{SLINK}^i(LCA(v, v'))) \\
&= i + \text{SDEP}(LCA(\text{SLINK}^i(v), \text{SLINK}^i(v'))) \\
&\geq i + \text{SDEP}(\text{LCSA}(\text{SLINK}^i(v), \text{SLINK}^i(v'))) \\
\end{align*}
\]

The last inequality is an equality for some \( i \leq d \).
With the previous lemma FCST’s compute the following operations:

- \( \text{SDEP}(v) = \text{SDEP}(\text{LCA}(v, v)) = \max_{0 \leq i < d} \{ i + \text{SDEP}(\text{LCSA}(\psi^i(v_l), \psi^i(v_r))) \} \).

- \( \text{LCA}(v, v') = \text{LF}(v[0..i - 1], \text{LCSA}(\psi^i(\min\{v_l, v'_l\}), \psi^i(\max\{v_r, v'_r\}))) \), for the \( i \) in the lemma.

- \( \text{SLINK}(v) = \text{LCA}(\psi(v_l), \psi(v_r)) \)
Problem (FCST’s are static)

How to insert or remove a text $T$ from a FCST that is indexing a collection $C$ of texts?

- Use Weiner’s algorithm or delete suffixes from the largest to the biggest.
- Update the CSA and the sampling.
Problem (FCST’s are static)

How to insert or remove a text $T$ from a FCST that is indexing a collection $C$ of texts?

- Use Weiner’s algorithm or delete suffixes from the largest to the biggest.
- Update the CSA and the sampling.
Problem (FCST’s are static)

How to insert or remove a text $T$ from a FCST that is indexing a collection $C$ of texts?

- Use Weiner’s algorithm or delete suffixes from the largest to the biggest.
- Update the CSA and the sampling.
Dynamic FCST’s

Use a dynamic CSA’s.

Theorem (Mäkinen, Navarro)

A dynamic CSA over a collection $\mathcal{C}$ can be stored in $nH_k(\mathcal{C}) + o(n \log \sigma)$ bits, with times $t = \Psi = O(((\log \sigma \log n)^{-1} + 1) \log n)$, $\Phi = O((\log \sigma \log n) \log^2 n)$, and inserting/deleting texts $T$ in $O(|T|(t + \Psi))$.

Let's take a closer look at the sampling.
Use a dynamic CSA’s.

**Theorem (Mäkinen, Navarro)**

*A dynamic CSA over a collection $\mathcal{C}$ can be stored in $nH_k(\mathcal{C}) + o(n \log \sigma)$ bits, with times $t = \Psi = O(((\log_\sigma \log n)^{-1} + 1) \log n)$, $\Phi = O((\log_\sigma \log n) \log^2 n)$, and inserting/deleting texts $T$ in $O(|T|(t + \Psi))$.*

Let’s take a closer look at the sampling.
How do we guarantee the sampling condition, with at most $O(n/\delta)$ nodes?

We use a purely conceptual reverse tree.

**Definition**

The reverse tree $T^R$ is the minimal labeled tree that, for every node $v$ of a suffix tree, contains a node $v^R$ denoting the reverse string of the path-label of $v$. 
How do we guarantee the sampling condition, with at most $O(n/\delta)$ nodes?

We use a purely conceptual reverse tree.

**Definition**

The reverse tree $T^R$ is the minimal labeled tree that, for every node $v$ of a suffix tree, contains a node $v^R$ denoting the reverse string of the path-label of $v$. 
How do we guarantee the sampling condition, with at most $O(n/\delta)$ nodes?

- We use a purely conceptual reverse tree.

**Definition**

The **reverse tree** $T^R$ is the minimal labeled tree that, for every node $v$ of a suffix tree, contains a node $v^R$ denoting the reverse string of the path-label of $v$.
Example (Suffix Tree for *abbbab* and its reverse tree)
Note that the SLINK’s correspond to moving upwards on the reverse tree.
We sample the nodes for which \( \text{TDEP}(v^R) \equiv \frac{\delta}{2} \) 0 and \( \text{HEIGHT}(v^R) \geq \frac{\delta}{2} \).
Example (Suffix Tree for `abbbab` and its reverse tree)

What happens when nodes are inserted or deleted?
Example (Suffix Tree for \textit{abbbab} and its reverse tree)

Only the leaves of the reverse tree change.
Example (Suffix Tree for \textit{abbbab} and its reverse tree)

This sampling does not respect the \textup{HEIGHT}(v^R) \geq \delta/2 condition.
To insert a node we do an upwards scan and sample nodes if necessary.
To insert a node we do an upwards scan and sample nodes if necessary.
To insert a node we do an upwards scan and sample nodes if necessary.
To delete a node we keep reference counters to guarantee that it is safe to unsample a node.
To delete a node we keep reference counters to guarantee that it is safe to unsample a node.
Other contributions

- We study the problem of a changing $\lceil \log n \rceil$.
- We give a new way to compute LSA.
- We obtain a generalized branching, that determines $v_1 \cdot v_2$ for nodes $v_1$ and $v_2$ and can be computed directly over CSA’s in the sample time as regular branching.
We study the problem of a changing $\lceil \log n \rceil$.

We give a new way to compute LSA.

We obtain a generalized branching, that determines $v_1 \cdot v_2$ for nodes $v_1$ and $v_2$ and can be computed directly over CSA's in the sample time as regular branching.
Other contributions

- We study the problem of a changing $\lceil \log n \rceil$.
- We give a new way to compute LSA.
- We obtain a generalized branching, that determines $v_1 \cdot v_2$ for nodes $v_1$ and $v_2$ and can be computed directly over CSA's in the sample time as regular branching.
We presented dynamic fully-compressed suffix trees that:

- occupy $uH_k + o(u \log \sigma)$ bits.
- supports usual operations in a reasonable time.
Veli Mäkinen and Johannes Fisher for pointing out the generalized branching problem.

FCT grant SFRH/BPD/34373/2006 and project ARN, PTDC/EIA/67722/2006.

Millennium Institute for Cell Dynamics and Biotechnology, Grant ICM P05-001-F, Mideplan, Chile.
Acknowledgments

Thanks for listening.