Finding Patterns in Trees and Strings

Philip Bille
Agenda

• Background for PhD

• Tree Matching
  • Tree Inclusion Problem

• String Matching
  • Regular Expression Matching Problem

• Core Techniques and Future Research

• Short Break

• Question Session
Background For PhD

• Worked on data structures.
  

• PhD funded by EU-project “Deep Structure, Singularities, and Computer Vision” working on tree matching problems.

• The Tree Inclusion Problem: In Optimal Space and Faster. Philip and Inge Li Gørtz. ICALP 2005.


• From a 2D Shape to a String Structure using the Symmetry Set. Arjan Kuijper, Ole Fogh Olsen, Peter Giblin, Philip Bille, and Mads Nielsen. ECCV 2004.

Basic Setup

• Trees are rooted, labeled, and ordered.

  • Rooted: A specific node is designated to be the root.

  • Labeled: Each node is assigned a label from an alphabet \( \Sigma \).

  • Ordered: There is a given left-to-right ordering among siblings.

• We compare trees by deleting nodes.
Deleting a Node
Deleting a Node
Deleting a Node
Deleting a Node
Tree Inclusion

- $P$ is *included* in $T$ if $P$ can be obtained from $T$ by deleting nodes in $T$.

- $P$ is *minimally included* in $T$ if $P$ is not included in any proper subtree of $T$.

- The *tree inclusion problem* is to decide if $P$ is included in $T$, and if so, compute all subtrees of $T$ which minimally include $P$. 
Example
Example
Example
Query: “Find all books written by Muthukrishnan with a chapter that has something to do with sampling”.
Query: “Find all books written by Muthukrishnan with a chapter that has something to do with sampling”.
## Results

<table>
<thead>
<tr>
<th>Time</th>
<th>Space</th>
<th>Ref</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(n_P n_T)$</td>
<td>$O(n_P n_T)$</td>
<td>[KM92]</td>
</tr>
<tr>
<td>$O(l_P n_T)$</td>
<td>$O(l_P \min(d_T, l_T))$</td>
<td>[Che98]</td>
</tr>
<tr>
<td>$O(l_P n_T)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$O(n_P l_T \log \log n_T + n_T)$</td>
<td>$O(n_P + n_T)$</td>
<td>[Here]</td>
</tr>
<tr>
<td>$O\left(\frac{n_P n_T}{\log n_T} + n_T \log n_T\right)$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Practical Implications

- Significant space reduction:
  - Feasible to query large XML databases.
  - Faster query time since more computation can be kept in main memory.
Algorithm Overview

• Reduce tree inclusion to tree embedding.

• Compute tree embeddings using a simple general framework.

• Implement the framework in 3 different ways to get the results.
Tree Inclusion and Embeddings

• An injective function $f$ from the nodes of $P$ to the nodes of $T$ is an embedding if for all nodes $v$ and $w$:

1. $\text{label}(v) = \text{label}(f(v))$,

2. $v$ is a proper ancestor of $w$ if and only if $f(v)$ is a proper ancestor of $f(w)$,

3. $v$ is to the left of $w$ if and only if $f(v)$ is to the left of $f(w)$.

• $P$ is included in $T$ if and only if there is an embedding from $P$ to $T$.

• $P$ is minimally included in $T$ if and only if there is an embedding from $P$ to $T$ and $P$ cannot be embedded in a proper subtree of $T$. 
Computing Embeddings: P is a Path
Computing Embeddings: P is a Path

Diagram:

- P is a simple path with nodes a and b.
- T is a tree with nodes labeled from a to b and c.
- The diagram illustrates how P is embedded into T.
Computing Embeddings: P is a Path

\[ P = \text{Active set} \]
Computing Embeddings: P is a Path

P = Active set
T = Root of min. subtree including
Computing Embeddings: P is a Path

P

T

○ = Active set  ○ = Root of min. subtree including
Computing Embeddings: P is a Path

P

T

○ = Active set
○ = Root of min. subtree including
Computing Embeddings: P is a Path

\( P = \text{Active set} \)

\( T = \text{Root of min. subtree including} \)
Computing Embeddings: P is a Path

P

T

= Active set

= Root of min. subtree including
Computing Embeddings: P is a Path

P

T

= Active set  = Root of min. subtree including
Computing Embeddings: P is a Path

P

T

○ = Active set ○ = Root of min. subtree including
Computing Embeddings: P is a Path

- **P** represents a path in the graph.
- **T** is the tree structure.
- The nodes in **T** are labeled with letters (a, b, c).
- The active set is marked with a blue circle.
- The root of the minimum subtree is marked with a red circle.

**Legend:**
- **Active set**
- **Root of min. subtree including**
Computing Embeddings: P is a Path

P

T

= Active set

= Root of min. subtree including
Computing Embeddings: P is a Path

P = Active set
T = Root of min. subtree including
Time Complexity

• At each step of the algorithm the active set “moves up”.

• Each parent pointer in \( T \) is traversed a constant number of times.

• Using a simple data structure and exploiting the ordering of the nodes we get a total running time of \( O(n_T) \).
Computing Embeddings: P is not a Path
Computing Embeddings: P is not a Path
Computing Embeddings: P is not a Path
Computing Embeddings: P is not a Path
Computing Embeddings: P is not a Path

P

T

b

a

b

a

b

a

b

a

b

a
Computing Embeddings: P is not a Path
Computing Embeddings: P is not a Path
Computing Embeddings: P is not a Path
Computing Embeddings: P is not a Path

P

T
Time Complexity

• Time complexity is bounded by the time used to compute embeddings for each root-to-leaf path in $P$.

• $\Rightarrow$ Time: $O(l_P n_T)$
Algorithm 2

- Reconsider the case when $P$ is path:

- Let $\text{firstlabel}(v, l)$ denote the nearest ancestor of node $v$ in $T$ labeled $l$.

- At each step we “essentially” compute $\text{firstlabel}(v, l)$ for each node $v$ in the active set.
Algorithm 2

- Idea: Use a fast data structure supporting first label queries. Known as the tree color problem.

- Theorem [Dietz1989]: For any tree $T$ there is a data structure using $O(n_T)$ space, $O(n_T)$ expected preprocessing time which supports first label queries in time $O(\log \log n_T)$. 
Time Complexity

• For each node in $P$ we have an active set of size at most $l_T$ and for each node in this active set we have to compute a first label query.

• => Time: $O(n_P l_T \log \log n_T + n_T)$
Algorithm 3: Idea

• Divide $T$ into $O(n_T / \log n_T)$ micro trees of size $O(\log n_T)$ which overlap in at most 2 nodes. Based on clustering technique from [AHLT1997].

• We represent each micro tree by a constant number of nodes in a macro tree and connect them according to the overlap of the micro trees.
Algorithm 3: Idea

- Active sets are represented compactly in $O(n_T / \log n_T)$ space as small bit strings for each micro tree.

- We preprocess micro trees using a “Four Russian Technique” such that we can update the active set in constant time for each micro tree.

- Leads to an $O\left(\frac{n_P n_T}{\log n_T} + n_T \log n_T\right)$ time algorithm.
Space Complexity

• Linear Space?

• No!
The Problem: Algorithm 1 and 2

- Storing all active sets uses $\Omega(l_T d_P)$ space.
Trick 1: Recurse to subtree with the most leaves

- The number of active sets stored does not exceed $O(\log l_P)$.

- $\Rightarrow$ Total space for stored active sets is $O(l_T \log l_P)$. 

\begin{itemize}
  \item $P$
  \item $l_T$
  \item $l_T$
  \item $l_T$
\end{itemize}
Trick 2: Strengthen Analysis

• Nodes in the active set for \( v \) are roots of (disjoint) subtrees that embed \( P(v) \).

• \( \Rightarrow \) Each of these subtrees have at least \( l_{P(v)} \) leaves.

• \( \Rightarrow \) The size of the active set for \( v \) is at most \( O(l_T/l_{P(v)}) \).
Space Complexity: Algorithm 1 and 2

• Trick 1 and 2 combined gives exponentially decreasing sizes of the stored active sets.

• => Total size of the stored active sets is $O(l_T)$.

• Space complexity is $O(n_P + n_T)$.

• Trick 2 shows that algorithm 2 in fact runs in $O(l_P l_T \log \log n_T + n_T)$ time.
Space Complexity: Algorithm 3

- Each active sets is represented in \( O(n_T / \log n_T) \) space.

- Trick 1 gives us that the total space for the stored active sets is

\[
O \left( \frac{n_T}{\log n_T} \log l_P \right) = O(n_T)
\]
Summary

• Time:

\[
\min \begin{cases} 
O(l_P n_T), \\
O(l_P l_T \log \log n_T + n_T), \\
O\left(\frac{n_P n_T}{\log n_T} + n_T \log n_T\right).
\end{cases}
\]

• Space: \(O(n_P + n_T)\)
String Matching


Regular Expressions

• The **regular expressions** are defined recursively:

• A character $\alpha \in \Sigma$ is a regular expression.

• If $S$ and $T$ are regular expressions then so is
  
  • the *concatenation* $ST$,

  • the *union* $S | T$, and

  • the *kleene star* $S^*$. 
Regular Expressions

• The language $L(R)$ of a regular expression $R$ is defined by:

• For any $\alpha \in \Sigma$, $L(\alpha) = \{\alpha\}$.

• For regular expressions $S$ and $T$:

$$L(ST) = L(S)L(T)$$

$$L(S|T) = L(S) \cup L(T)$$

$$L(S^*) = \{\epsilon\} \cup L(S) \cup L(S)^2 \cup L(S)^3 \cup \ldots$$
Example

\[ R = ac | a^*b \]

\[ L(R) = \{ac, b, ab, aab, aaab, aaaaab, \ldots \} \]
Regular expression Matching

• Given a regular expression $R$ and a string $Q$ the regular expression matching problem is to decide if $Q \in L(R)$.

• Example: $R = ac | a^*b$ matches $Q = aaaaab$. 
Applications:

- Lexical analysis phase in compilers.
- Protein searching.
- Text editing and programming languages (e.g. EMACS and Perl).
## Results

<table>
<thead>
<tr>
<th>Time</th>
<th>Space</th>
<th>Ref</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(nm)$</td>
<td>$O(m)$</td>
<td>[Tho68]</td>
</tr>
<tr>
<td>$O((n + 2^m)\lceil m/w \rceil)$</td>
<td>$O((2^m + \sigma)\lceil m/w \rceil)$</td>
<td>[NR04]</td>
</tr>
<tr>
<td>$O\left(\frac{nm}{\log n} + n + m \log m\right)$</td>
<td>$O(n)$</td>
<td>[Mye92, Here]</td>
</tr>
<tr>
<td>(\begin{cases} O\left(n\frac{m \log w}{w} + m \log w\right) &amp; \text{if } m &gt; w \ O(n \log m + m \log m) &amp; \text{if } \sqrt{w} &lt; m \leq w \ O(\min(n + m^2, n \log m + m \log m)) &amp; \text{if } m \leq \sqrt{w} \end{cases})</td>
<td>$O(m)$</td>
<td>[Here]</td>
</tr>
</tbody>
</table>
Practical Implications

• Except for Thompson’s algorithm all previous algorithms use large tables and perform a long series of lookups in the tables.

• => Many expensive cache misses.

• New algorithm does not require the large tables.
Algorithm Overview

- Construct non-deterministic finite automata (NFA) using Thompson’s classical algorithm.

- Decompose the NFA into small subautomata.

- Simulate each subautomata using the arithmetic and logical instruction of the word RAM.

- Use the simulation of the each subautomata to simulate the entire NFA.
Thompson’s Algorithm

\[ N(\alpha) \]

\[ N(S|T) \]

\[ N(ST) \]

\[ N(S^*) \]
Thompson NFA

- Thompson-NFA (TNFA) for $R = ac|a^*b$.

- $N(R)$ accepts $Q$ if and only if there is path from $\theta$ to $\phi$ that “spells” out $Q$.

- $Q \in L(R)$ if and only if $N(R)$ accepts $Q$. 
Properties of TNFAs

• Linear number of states and transitions.

• Incoming transitions to a state have the same label.

• States with an incoming transition labeled $\alpha \in \Sigma$ (\(\alpha\)-states) have exactly 1 predecessor.
Simulating TNFAs

- Let $A$ be TNFA with $m$ states. To test acceptance we use the following operations. For a state-set $S$ and $\alpha \in \Sigma$:

  - $\text{Move}(S, \alpha)$: Find set of states reachable from $S$ via a single $\alpha$-transition.

  - $\text{Close}(S)$: Find set of states reachable from $S$ via a path of $\varepsilon$-transitions.

  - $O(m)$ time for both operations.
Simulating TNFAs

• Let \( Q \) be a string of length \( n \).

• The state-set simulation of \( A \) on \( Q \) produces state-sets \( S_0, S_1, \ldots, S_n \) as follows:

\[
S_0 := \text{Close}(\{\theta\})
\]

\[
S_i := \text{Close}(\text{Move}(S_{i-1}, Q[i]))
\]

• \( S_i \) is the set of states reachable from \( \theta \) through a path that spells out \( Q[1..i] \).

• \( Q \in L(R) \) if and only if \( \phi \in S_n \).

• \( O(nm) \) time and \( O(m) \) space.
Four-Russian Speedup

• Decompose TNFA into sub-automata with $O(\log n)$ states.

• Preprocess subautomata to get Move and Close in constant time for each. Subautomata are made “deterministic”.

• $\Rightarrow O \left( \frac{nm}{\log n} + n + m \log m \right)$ time and $O(n)$ space algorithm [Myers92, BFC05].
Word-Level Parallel Algorithm

- Idea: Use essentially same decomposition into subautomata.

- Simulate Move and Close using the arithmetic and logical instructions of the word RAM.
Simple Algorithm for small TNFAs

• Suppose \( A \) is a TNFA with \( m = O(\sqrt{w}) \) states.

• Order the states such that the (unique) predecessor of \( \alpha \)-state \( i \) is \( i - 1 \).

• Represent state-sets as a bit string.
Representation of State-Sets

\[ S = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \]
Move Operation: Preprocessing

- For each $\alpha \in \Sigma$ represent $\alpha$-states using a bit string:

$D_a = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$

$D_b = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$
Move Operation: Simulation

- We compute \( \text{Move}(S, \alpha) \) as

\[
S' := (S >> 1) \& D_\alpha
\]
Example: $S' := (S >> 1) \& D_\alpha$

- Compute $\text{Move}(S, a)$ where $S = \begin{array}{c}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
0 & 1 & 1 & 0 & 1 & 0 & 0 & 0
\end{array}$:

$S >> 1 = \begin{array}{c}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
0 & 0 & 1 & 1 & 0 & 1 & 0 & 0
\end{array}$

& $D_a = \begin{array}{c}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0
\end{array}$

$\overline{0 0 0 1 0 0 0 0}$
Example: $S' := (S >> 1) \& D_\alpha$

- Compute $\text{Move}(S, a)$ where $S = 011010000$:

  
  \[
  S >> 1 \quad 001101000 \\
  \& D_a \quad 000100000 \\
  \hline
  000100000
  \]

  
  

Example: $S' := (S >> 1) \& D_\alpha$

- Compute $\text{Move}(S, a)$ where $S = \begin{array}{c} 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \end{array}$:

\[
S >> 1 = \begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
0 & 0 & 1 & 1 & 0 & 1 & 0 & 0
\end{array}
\]

& $D_a = \begin{array}{cccccccc}
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0
\end{array}$
Example: $S' := (S \gg 1) \& D_\alpha$

- Compute $\text{Move}(S, a)$ where $S = \begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
0 & 1 & 1 & 0 & 1 & 0 & 0 & 0
\end{array}$:

<table>
<thead>
<tr>
<th>$S \gg 1$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

$\& D_a$

| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
Close Operation: Preprocessing

• Encode $\epsilon$-paths compactly:

$$E = \begin{bmatrix} 0 & E_1 & 0 & E_2 & 0 & E_3 & 0 & E_4 & 0 & E_5 & 0 & E_6 & 0 & E_7 & 0 & E_8 \end{bmatrix}$$

$$E_2 = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$
Close Operation: Preprocessing

- 3 constant bit strings for doing word tricks:

\[ I = (10^m)^m \]
\[ X = 1(0^m)^{m-1} \]
\[ C = 1(0^{m-1})^{m-1} \]
Close Operation: Simulation

- Close($S$) is computed as:

\[
Y := (S \times X) \& E \\
Z := ((Y \mid I) - (I >> m)) \& I \\
S' := ((Z \times C) << w - m(m + 1)) >> w - m
\]
Example:

• Compute $\text{Close}(S)$ where $S = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}$. 
Step 1: $Y := (S \times X) \& E$

$$S \times X = \begin{bmatrix} 0 & S & 0 & S & 0 & S & 0 & S & 0 & S & 0 & S \\ \end{bmatrix}$$

$$&E = \begin{bmatrix} 0 & E_1 & 0 & E_2 & 0 & E_3 & 0 & E_4 & 0 & E_5 & 0 & E_6 & 0 & E_7 & 0 & E_8 \\ \end{bmatrix}$$

$$Y = \begin{bmatrix} 0 & Y_1 & 0 & Y_2 & 0 & Y_3 & 0 & Y_4 & 0 & Y_5 & 0 & Y_6 & 0 & Y_7 & 0 & Y_8 \\ \end{bmatrix}$$

$$S = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ \end{bmatrix}$$

$$&E_2 = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ \end{bmatrix}$$

$$Y_2 = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ \end{bmatrix}$$
Step 2: $Z := ((Y | I) - (I >> m)) \& I$

| $Y | I$ | 1 | $Y_1$ | 1 | $Y_2$ | 1 | $Y_3$ | 1 | $Y_4$ | 1 | $Y_5$ | 1 | $Y_6$ | 1 | $Y_7$ | 1 | $Y_8$ |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| $-(I >> m)$ | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| $\& I$ | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| $Z = 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |

$Y_2 = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ -0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$
Step 3: $S' := ((Z \times C) \ll \ d m(m + 1)) \gg \ d m$

- $Z \times C$ produces a bit string containing the test bits of $Z$ as a consecutive substring.

- Shift clears remaining bits and aligns the substring.
Complexity

• Lemma: For TNFAs with $O(\sqrt{w})$ states we can support Move and Close in constant time using $O(m)$ space and $O(m^2)$ preprocessing.

• $\Rightarrow$ For string $Q$ and regular expression $R$ of lengths $n$ and $m = O(\sqrt{w})$ regular expression matching can be solved in $O(n + m^2)$ time and $O(m)$ space.
Another Algorithm

- Main bottleneck: Need an $\Omega(m^2)$ length string to represent the transitive closure of $\epsilon$-transitions.

- Idea: Compute a “good” separator for TNFAs and use a Divide-and-Conquer strategy.
There exists two states $\theta_{P_i}$ and $\phi_{P_i}$ whose removal partitions a TNFA into two subgraphs, $P_i$ and $P_O$, of roughly equal size such that:

- Any path from $P_O$ to $P_i$ goes through $\theta_{P_i}$.
- Any path from $P_i$ to $P_O$ goes through $\phi_{P_i}$.
Recursive Closure Algorithm

1. Determine which of $\theta_{P_i}$ and $\phi_{P_i}$ are $\epsilon$-reachable.

2. Update the state-set accordingly.

3. Recurse in parallel on $P_i$ and $P_O$.
Complexity

- Each of the $O(\log m)$ levels of recursion can be handled in parallel in constant time.

- $\Rightarrow$ Lemma: For TNFAs with $m = O(w)$ states we can support Move and Close in $O(\log m)$ time using $O(m)$ space and $O(m \log m)$ preprocessing.

- $\Rightarrow$ For string $Q$ and regular expression $R$ of lengths $n$ and $m = O(w)$, resp., regular expression matching can be solved in time $O(n \log m + m \log m)$ and space $O(m)$. 
Plug and Play

- **Time:**

\[
\begin{cases}
O(n \frac{m \log w}{w} + m \log w) & \text{if } m > w \\
O(n \log m + m \log m) & \text{if } \sqrt{w} < m \leq w \\
O(\min(n + m^2, n \log m + m \log m)) & \text{if } m \leq \sqrt{w}.
\end{cases}
\]

- **Space:** \(O(m)\)
Core Techniques

• Data Structures: Organize information efficiently.
  • Nearest common ancestors, first label, dictionaries, dynamic perfect hashing, predecessors.

• Tree Techniques: Use combinatorial properties of trees.
  • Heavy-path decomposition, varieties of tree clusterings with or without macro trees.

• Word-Level Parallelism: Encode and simulate algorithms using arithmetic and logical instructions of the word RAM.
  • Four Russian technique, word level-parallelism.
Future Research

• Bring state-of-the-art techniques to combinatorial pattern matching and related areas. Many important problems need them!

• Use developed algorithms to improve practical applications (e.g., bioinformatics, XML data bases).

  • Word parallel regular expression matching looks promising.
Thanks!