Faster Bit-Parallel Algorithms for Unordered Pseudo-Tree Matching and Tree Homeomorphism

Yusaku Kaneta and Hiroki Arimura

Graduate School of Information Sci. and Tech. Hokkaido University, Japan
Background: Tree matching problem

- Problem of finding an embedding $\phi$ from a pattern tree $P$ to a text tree $T$
- Fundamental problem in computer science [Kilpelainen & Mannila, ‘94]
- It has many applications
- We consider unordered tree matching and its variants (for labeled, rooted tree)

Embedding $\phi$

Pattern tree $P$

Text tree $T$

(E0) $\phi$ is one-to-one
(E1) $\phi$ preserves the node labels
(E2) $\phi$ preserves the parent-child relation
Background: Many-to-one matching

- In original theoretical studies: Tree matching with one-to-one mapping has been mainly studied so far.
- In recent practical studies: Tree matching with many-to-one mapping attracts much attention.
- **Goal:** To develop efficient algorithms for two tree matching problems with many-to-one mappings.
  - Unordered pseudo-tree matching problem (UPTM) \iff XPath queries with child axis only
  - Unordered tree homeomorphism problem (UTH) \iff XPath queries with descendant axis only
Definition

- **Unordered pseudo-tree matching problem (UPTM)**
  - A pattern tree $P$ matches a text tree $T$ if there is a many-to-one mapping $\phi: V(P) \rightarrow V(T)$ from $P$ into $T$ satisfying the conditions (E1) and (E2)
  - An occurrence of $P$ in $T$ is the image of the root of $P$
  - The problem is to find all occurrences of $P$ in $T$

- **Unordered tree homeomorphism problem (UTH)**
  - is defined similarly, where many-to-one mapping satisfying (E1) and (E3) is used.

$\phi$ preserves:
- (E1) the node labels
- (E2) the parent-child relation
- (E3) the ancestor-descendant relation
Related work

● Many studies for tree matching with one-to-one mappings
  - [Kilpelainen, Mannila, SIAM J’95]: The unordered tree matching and inclusion problems
  - Corresponds to the subgraph isomorphism problem

● Few studies for tree matching with many-to-one mappings
  - [Yamamoto, Takenouchi, WADS’09] UPTM problem
    - \( O(nr \cdot \text{leaves}(P) \cdot \text{depth}(P)/w) = O(nm^3/w) \) time
    - \( O(n \cdot \text{leaves}(P) \cdot \text{depth}(P)/w) = O(nm^2/w) \) space
  - [Gotz, Koch, Martens, DBPL’07] UTH problem
    - \( O(nm \cdot \text{depth}(P)) = O(nm^2) \) time
    - \( O(\text{depth}(T) \cdot \text{branch}(T)) = O(n^2) \) space

\( m \): the size of \( P \), \( n \): the size of \( T \), \( h \): the height of \( T \), \( w \): the word length, and \( r \): the maximum number of the same label on paths in \( P \)
Our results

- **New decomposition formula** for unordered pseudo-tree matching problem (UPTM)
- **Bit-parallel algorithm** for UPTM that runs in:
  - $O(nm\log(w)/w)$ time
  - $O(hm/w + m\log(w)/w)$ space
  - $O(m\log(w))$ preprocessing time
- **Key**: Fast bit-parallel computation of Tree aggregation in $O(\log m)$ time
  - Improves a naïve implementation in $O(m)$ time
- Modified algorithm for **UTH** with the same complexity

$m$: the size of $P$, $n$: the size of $T$, $h$: the height of $T$, $w$: the word length
## Summary

<table>
<thead>
<tr>
<th>Algorithm for UPTM</th>
<th>Time</th>
<th>Space (in words)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BP-MatchUPTM (this work)</strong></td>
<td>$O(nm\log(w)/w)$</td>
<td>$O(hm/w + m\log(w)/w)$</td>
</tr>
<tr>
<td>[Yamamoto, Takenouchi, WADS’09]</td>
<td>$O(nm^3/w)$</td>
<td>$O(nm^2/w)$</td>
</tr>
</tbody>
</table>

- Our algorithm improves the algorithm by [YT’09] (by $O(m^2/\log(w))$)

<table>
<thead>
<tr>
<th>Algorithm for UTH</th>
<th>Time</th>
<th>Space (in words)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BP-MatchUTH (this work)</strong></td>
<td>$O(nm\log(w)/w)$</td>
<td>$O(hm/w + m\log(w)/w)$</td>
</tr>
<tr>
<td>[Gotz, Koch, Martens, DBPL’07]</td>
<td>$O(nm^2)$</td>
<td>$O(hn)$</td>
</tr>
</tbody>
</table>

- Our algorithm improves the algorithm by [Gotz et al.’07]
- This is the first bit-parallel algorithm for UTH (by $O(mw/\log(w))$)

$m$: the size of $P$, $n$: the size of $T$, $h$: the height of $T$, $w$: the word length


Algorithm for the UPTM problem
Our algorithm MatchUPTM

Consists of two components:

1. **New decomposition formula** for bottom-up computation

2. **Bit-parallel implementation** of five set operations: Constant, Union, Member, LabelMatch, and TreeAggr

- Especially, $O(\log m)$ time bit-parallel implementation of TreeAggr operation
Decomposition formula for UPTM

- The embedding set $\text{Emb}^{P,T}(v)$ of text node $v \in V(T)$ is the set of pattern node $x \in V(P)$ such that $P(x)$, the subtree of $P$ rooted at $x$, occurs in $T$ at node $v$.

**Lemma 1 (decomposition formula):** For any $x \in V(P)$, $v \in V(T)$, $x \in \text{Emb}^{P,T}(v)$

$\iff$ (i) Label matching: $\text{label}_P(x) = \text{label}_T(v)$ and
(ii) Tree aggregation: $\text{children}(x) \subseteq \bigcup_{1 \leq j \leq \alpha(v)} \text{Emb}^{P,T}(v[j])$

- From Lemma 1, we can develop a bottom-up algorithm for UPTM in $O(nm)$ time and $O(hm)$ space, where $h$ is the height of $T$.

![Pattern tree $P$ and text tree $T$](image)
**Bit-parallel implementation**

- To obtain further speed-up, we use **bit-parallelism**
  - Encoding an embedding set $\text{Emb}(v) \subseteq \{1,\ldots,m\}$ for each node $v$ by a bitmask $X \in \{0,1\}^m$ of length $m$.
  - By implementing the five set operations by using Bit-wise Boolean operations $\&$, $|$, $\sim$ and integer addition $+$ [BGY’92]

- **Key:** Bit-parallel implementation of $\text{TreeAggr}_p$

<table>
<thead>
<tr>
<th>Operation</th>
<th>Original impl.</th>
<th>Bit-parallel impl.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{Constant}(S)$</td>
<td>$O(m)$ time</td>
<td>$O(m/w)$ time</td>
</tr>
<tr>
<td>$\text{Union}(R, S)$</td>
<td>$O(m)$ time</td>
<td>$O(m/w)$ time</td>
</tr>
<tr>
<td>$\text{Member}(R, x)$</td>
<td>$O(m)$ time</td>
<td>$O(m/w)$ time</td>
</tr>
<tr>
<td>$\text{LabelMatch}_p(R, \alpha)$</td>
<td>$O(m)$ time</td>
<td>$O(m/w)$ time (From [BYG92])</td>
</tr>
<tr>
<td>$\text{TreeAggr}_p(R, S)$</td>
<td>$O(m)$ time</td>
<td>$O(m \log(w)/w)$ time (This work)</td>
</tr>
</tbody>
</table>


$m$: the size of $P$, $n$: the size of $T$, $w$: the word length
Bit-parallel tree aggregation

- Computes the parent value as the logical AND of the children values
- **Preprocess**: Build the following bitmasks
  - **DST**: the position of parent \( x \)
  - **SRC**: the positions of children \( \text{children}(x) \)
  - **SEED**: the lowest position of component \( C_x \)
  - **INT**: the interval of \( C_x \) except for \( x \) and \( \text{children}(x) \)
- **Runtime**: Simulate tree aggregation by bit-operations
Bit-parallel tree aggregation

- **Basic idea:** Using the carry propagation by integer addition
- **Line 2:** Compute the PATH mask. We fill the “holes” at the children positions in INT with the children values in the input mask Y.

```
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

COMP ← Y & SRC;
PATH ← COMP | INT;
AGGR ← PATH + SEED;
RESULT ← AGGR & DST;
```

- **Line 3:** Compute the AGGR mask. If all the “holes” in PATH are filled then the parent value is set.

```
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

PATH ← 0 1 1 1 1 1 1 1
SEED ← 0 0 0 0 0 0 0 0
AGGR ← 1 0 0 0 0 0 0 0
```

**Runtime:**
1. COMP ← Y & SRC;
2. PATH ← COMP | INT;
3. AGGR ← PATH + SEED;
4. RESULT ← AGGR & DST;
Separator tree-based decomposition

- By using the separator tree-based decomposition technique, we can implement Tree Aggregation in $O(\log(m))$ time using $O(m \log m)$ preprocessing time.

**Lemma (Jordan, 1869).** Let $S$ be a binary tree. Then, there exists a node in $S$ such that $|S(v)| \leq (2/3)|S|$ and $|S(v')| \leq (2/3)|S|$, where $S(v)$ is the subtree of $S$ rooted at $v$ and $S(v')$ is the tree obtained by pruning $S(v)$ from $S$.

Naïve decomposition:

<table>
<thead>
<tr>
<th>Bit-position</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Level 1</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>Level 3</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Level 4</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

Separator tree-based decomposition:

<table>
<thead>
<tr>
<th>Bit-position</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Level 1</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Level 3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>Level 4</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

O(m) vs. O(\log m)
Main result for the UPTM problem

- By applying the module decomposition techniques of [Myers ‘92] and [Bille ‘06], we have:

Theorem 1. (complexity of the UPTM problem)
The algorithm BP-MatchUPTM solves the unordered pseudo-tree matching problem in:
- $O(nm\log(w)/w)$ time, using
- $O(hm/w + m\log(w)/w)$ space and
- $O(m\log(w))$ preprocessing time

$m$: the size of P, $n$: the size of T, $h$: the height of T, $w$: the word length

Note: This improves the time complexity $O(nm^3/w)$ of the previous bit-parallel algorithm by [Yamamoto & Takenouchi, WADS’09] with a factor of $O(m^2/\log(w))$

Main result for the UTH problem

- Modified Bit-parallel algorithm BP-MatchUTH:
  - Based on a similar decomposition formula
  - The code is same as VisitUPTM except line 9

Theorem 2. (complexity of the UTH problem)
The algorithm BP-MatchUTH solves the unordered tree homeomorphism problem in
- $O(nm\log(w)/w)$ time
- $O(hm/w + m\log(w)/w)$ space
- $O(m\log(w))$ preprocessing time

$m$: the size of $P$, $n$: the size of $T$, $h$: the height of $T$, $w$: the word length

Note: This seems the first bit-parallel algorithm for UTH problem as far as we know, and it slightly improves the time complexity $O(nm^2)$ of the algorithm by [Gotz, Koch, Martens, DBPL'07] with a factor of $O(mw/\log(w))$
Conclusion

- Tree matching with many-to-one mapping
  - **UPTM**: unordered pseudo-tree matching
  - **UTH**: unordered tree homeomorphism

- Bit-parallel algorithms for **UPTM** and **UTH** that run in
  - $O(nm\log(w)/w)$ time
  - $O(hm/w + m\log(w)/w)$ space
  - $O(m\log(w))$ preprocessing

- Future works
  - Extension of this technique for tree matching and inclusion with one-to-one mappings (seems difficult)
  - Applications to practical subclasses of XPath and XQuery languages
Thank you