21st International Workshop on Combinatorial Algorithms (IWOCA'10)

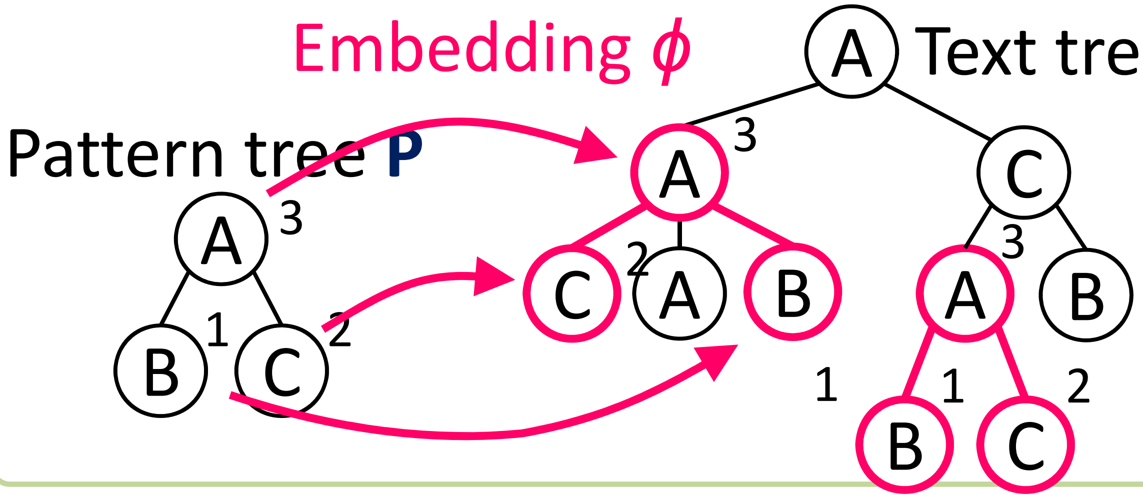
Faster Bit-Parallel Algorithms for Unordered Pseudo-Tree Matching and Tree Homeomorphism

Yusaku Kaneta and Hiroki Arimura

Graduate School of Information Sci. and Tech.
Hokkaido University, Japan

Background: Tree matching problem

- Problem of finding an embedding ϕ 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)



(E0) ϕ is one-to-one
(E1) ϕ preserves the node labels
(E2) ϕ preserves the parent-child relation

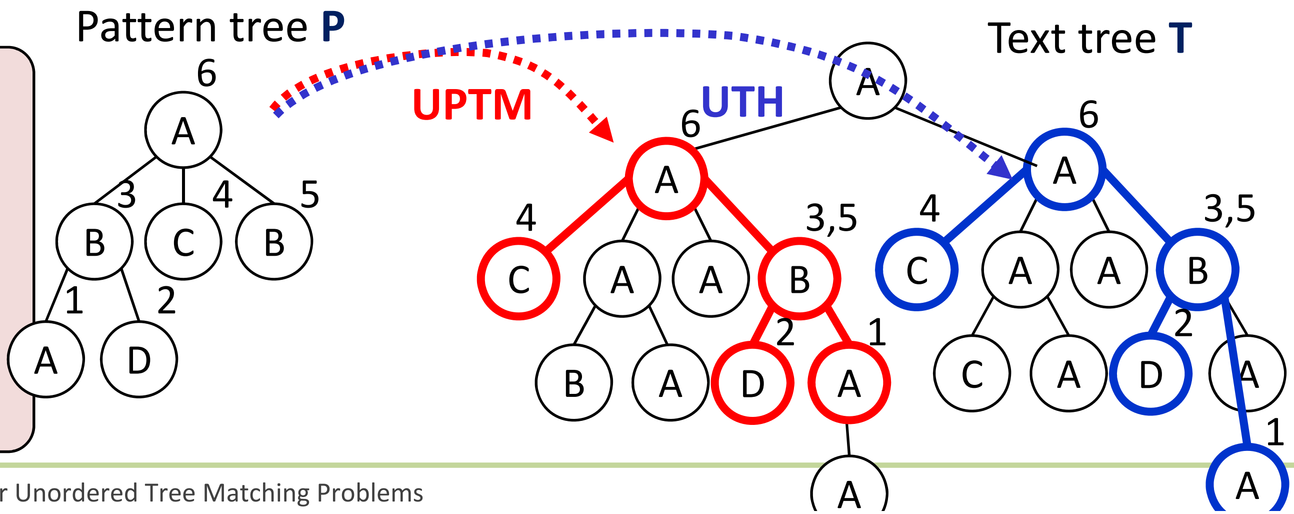
Background: Many-to-one matching

- In original theoretical studies:
  Tree matching with one-to-one mapping has been mainly studied so far
- In recent practical studies: Tree maching 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**)
    ⇔ XPath queries with child axis only
  - Unordered tree homeomorphism problem (**UTH**)
    ⇔ XPath queries with descendant axis only

Definition

- **Unorderd pseudo-tree matching problem (UPTM)**
  - A pattern tree **P** matches a text tree **T** if there is a many-to-one mapping ***ϕ*: V(P) → 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
- Unorderd tree homeomorphism problem (UTH)
  - is defined similarly, where many-to-one mapping satisfying **(E1)** and **(E3)** is used.

ϕ 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**
    - **$O(nr \cdot leaves(P) \cdot depth(P)/w) = O(nm^3/w)$** time
    - **$O(n \cdot leaves(P) \cdot depth(P)/w) = O(nm^2/w)$** space
  - **[Gotz, Koch, Martens, DBPL'07]** **UTH problem**
    - **$O(nm \cdot depth(P)) = O(nm^2)$** time
    - **$O(depth(T) \cdot branch(T)) = O(n^2)$** 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(nmlog(w)/w)** time
  - **O(hm/w + mlog(w)/w)** space
  - **O(mlog(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

Algorithm for UPTM	Time	Space (in words)
BP-MatchUPTM (this work)	$O(nm\log(w)/w)$	$O(hm/w + m\log(w)/w)$
[Yamamoto, Takenouchi, WADS'09]	$O(nm^3/w)$	$O(nm^2/w)$

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

Algorithm for UTH	Time	Space (in words)
BP-MatchUTH (this work)	$O(nm\log(w)/w)$	$O(hm/w + m\log(w)/w)$
[Gotz, Koch, Martens, DBPL'07]	$O(nm^2)$	$O(hn)$

- 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

[Yamamoto, Takenouchi, WADS'09] H. Yamamoto and D. Takenouch, Bit-parallel tree pattern matching algorithms for unordered labeled trees, In *Proc*. WADS'09, 554-565, 2009.

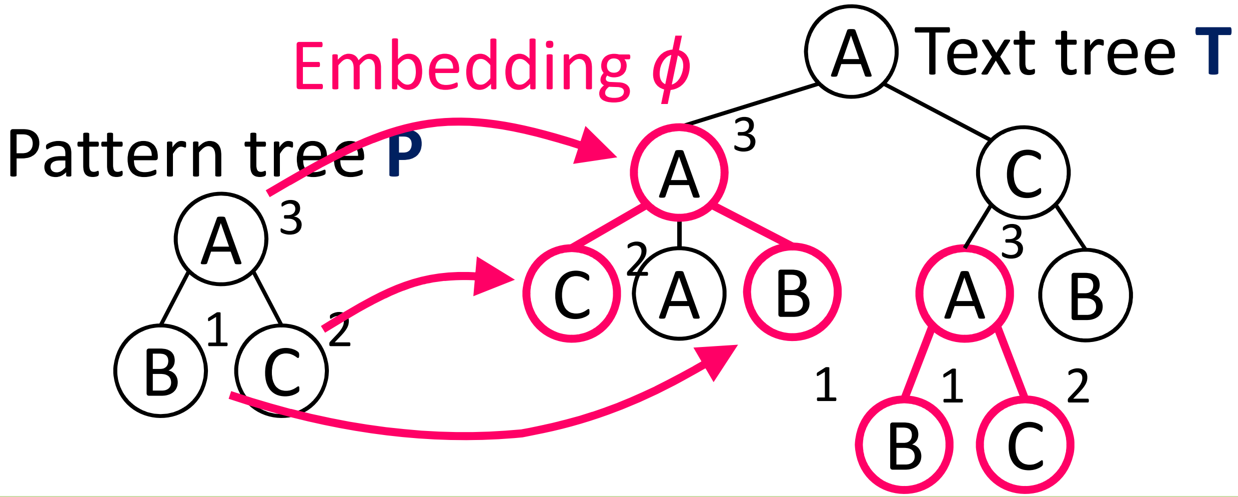
[Gotz, Koch, Martens, DBPL'07] M. Gotz, C. Koch, and W. Martens, Efficient algorithms for tree homeomorphism problem, In *Proc*. DBPL'07, 17-31, 2007.

Algorithm
for the UPTM problem

Our algorithm MatchUPTM

Consists of two components:

1. **New decomposition formula** for bottom-up computataion
2. **Bit-parallel implementation** of five set operations: **Constant, Union , Member , LabelMatch$_P$,** and **TreeAggr$_P$**
   - **Especiallly, O(log m)** time bit-parallel implementation of **TreeAggr operation**



Decomposition formula for UPTM

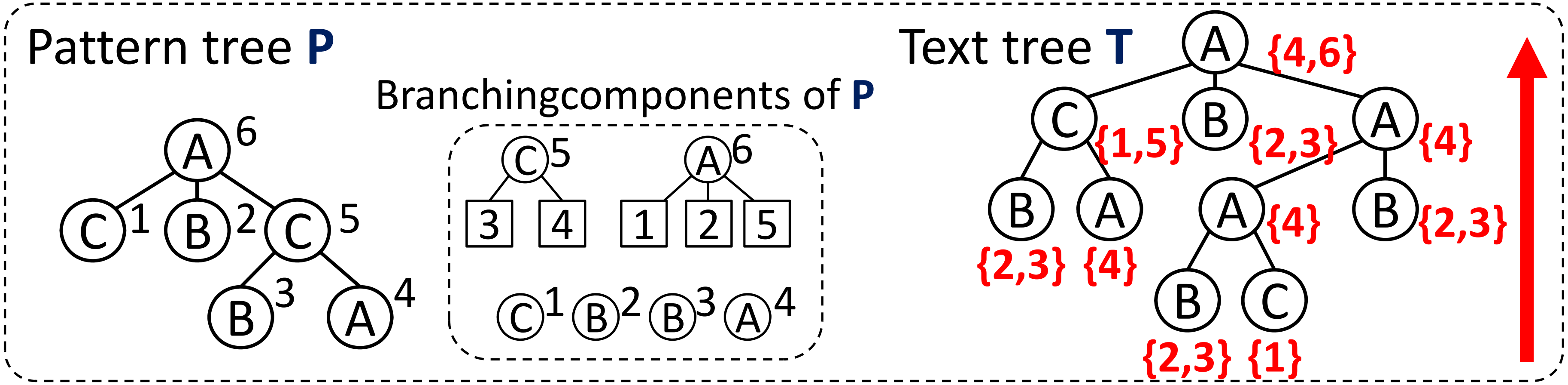
- The embedding set $\mathbf{Emb}^{P,T}(\mathbf{v})$ of text node $\mathbf{v} \in \mathbf{V(T)}$
  - is the set of pattern node $\mathbf{x} \in \mathbf{V(P)}$ such that P(x), the subtree of **P** rooted at **x**, occurs in **T** at node **v**

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

$\Leftrightarrow$ (i) Label matching: $\mathbf{label_P(x) = label_T(v)}$ and

(ii) Tree aggregation: $\mathbf{children(x)} \subseteq \bigcup_{1 \le j \le \alpha(v)} \mathbf{Emb}^{P,T}(\mathbf{v[j]})$

- From Lemma1, we can develop a bottom-up algorithm for UPTM in **O(nm)** time and **O(hm)** space, where **h** is the height of **T**



Bit-parallel implementation

- To obtain further speed-up, we use bit-parallelism
  - Encoding an embedding set **Emb(v)** $\subseteq$ {1,...,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 &, |, ~ and integer addition + [BGY'92]
- Key: Bit-parallel implementation of **TreeAggr$_P$**

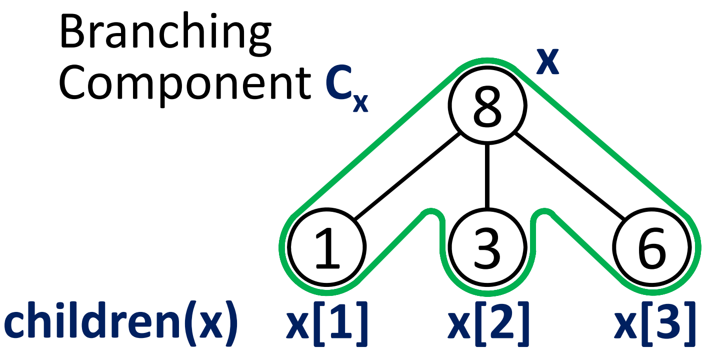
Operation	Original impl.	Bit-parallel impl.	
Constant(S)	**O(m)** time	**O(m/w)** time	Easy
Union(R, S)	**O(m)** time	**O(m/w)** time	
Member(R, x)	**O(m)** time	**O(m/w)** time	
LabelMatch$_P$(R, α)	**O(m)** time	**O(m/w)** time (From **[BYG92]**)	
TreeAggr$_P$(R, S)	**O(m)** time	**O(m log(w)/w)** time **(This work)**	**Hard to implement**

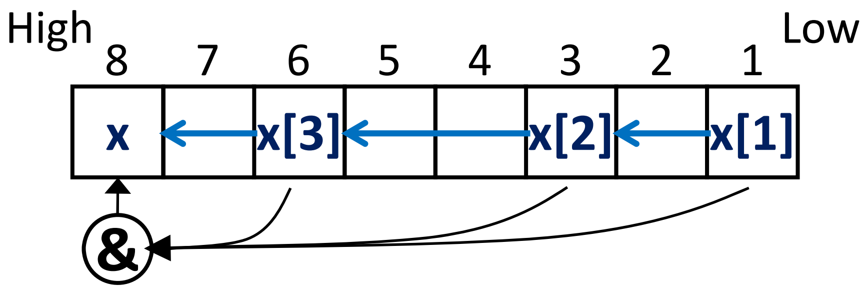
[BYG'92] R. Baeza-Yates and G. H. Gonnet, CACM, 35(10), 74-82, 1992.

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 **children(x)**
  - **SEED**: the lowest position of component $\mathbf{C_x}$
  - **INT**: the interval of $\mathbf{C_x}$ except for **x** and **children(x)**
- **Runtime**: Simulate tree aggregation by bit-operations





Branching omponent $\mathbf{C_x}$: the connected component consisting of parent **x** and its children **children(x)**

Bit-parallel tree aggregation

- **Basic idea: Using the carry propergation by integer addition**
- **Line2**: Compute the **PATH** mask. We fill the "holes" at the children positions in **INT** with the children values in the input mask **Y**.

	x		x[3]			x[2]		x[1]
COMP	0	0	1	0	0	1	0	1
|) INT	0	1	0	1	1	0	1	0
PATH	0	1	**1**	1	1	**1**	1	**1**

holes

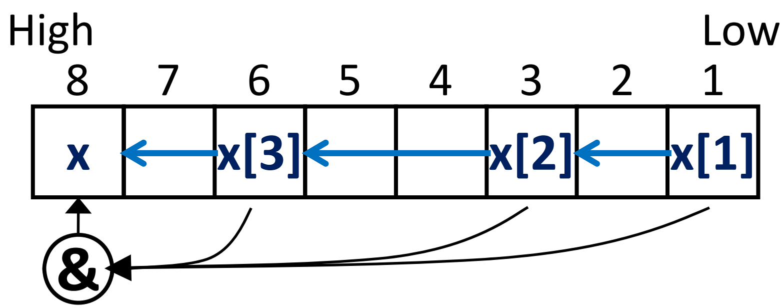
- **Line3**: Compute the **AGGR** mask. If all the "holes" in **PATH** are filled then the parent value is set

	x		x[3]			x[2]		x[1]
PATH	0	1	1	1	1	1	1	1
+) SEED	0	0	0	0	0	0	0	**1**
AGGR	**1**	0	0	0	0	0	0	0

carries

Runtime:

1. **COMP** ← **Y** & **SRC**;
2. **PATH** ← **COMP** | **INT** ;
3. **AGGR** ← **PATH** + **SEED**;
4. **RESULT** ← **AGGR** & **DST**;



Input:

	x		x[3]			x[2]		x[1]
Y	0	1	1	0	1	1	0	1

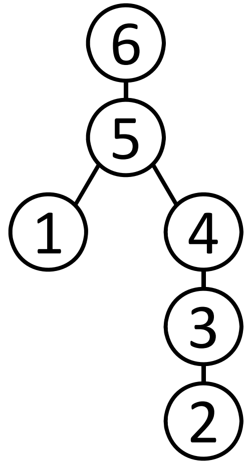
Bitmasks:

DST	**1**	0	0	0	0	0	0	0
SRC	0	0	**1**	0	0	**1**	0	**1**
SEED	0	0	0	0	0	0	0	**1**
INT	**0**	1	**0**	1	1	**0**	1	**0**

Separator tree-based decomposition

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

Pattern tree **P**



> **Lemma (Jordan , 1869).** Let **S** be a binary tree. Then, there exists a node in S such that **|S(v)| ≤ (2/3)|S|** and **|S(v')| ≤ (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:

Bit-position	1	2	3	4	5	6
Node	**1**	**2**	**3**	**4**	**5**	**6**
Level 1					5 →	6
Level 2	1 →			4 →	5	
Level 3			3 →	4		
Level 4		2 →	3			

O(m)

Separator tree-based decomposition:

Bit-position	1	2	3	4	5	6
Node	**2**	**3**	**4**	**1**	**5**	**6**
Level 1	1 →			4	5	
Level 2	2 →	3				
Level 3			3 →	4	5 →	6

O(log m)

Bit-assignment also differs from naïve decomposition.

Main result for the UPTM problem

- By appying 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(nmlog(w)/w)** time, using
- **O(hm/w + mlog(w)/w)** space and
- **O(mlog(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 **$O(m^2/\log(w))$**

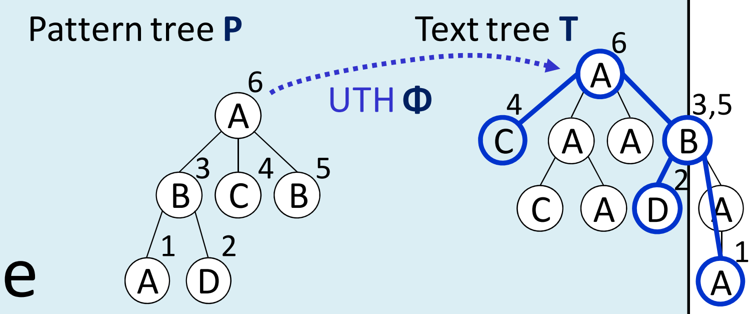
[Bille'06] P. Bille, New algorithms for regular expression matching, In *Proc*. ICALP'06, 643-654, 2006.
[Myers'92] E. W. Myers, A four-russian algorithm for regular expression pattern matching, JACM, 39(2), 430-448, 1992.

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(nmlog(w)/w)** time
- **O(hm/w + mlog(w)/w)** space
- **O(mlog(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(nm²)** 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(nmlog(w)/w)** time
  - **O(hm/w + mlog(w)/w)** space
  - **O(mlog(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