Discovering Frequent Substructures in Large Unordered Trees Unot

WS08/09

Seminar aus Maschinellem Lernen

Prof. J. Fürnkranz

Fan Zhang

Inhalt

- Motivation
- Basic Definitions
- Canonical Representation
- Algorithm Unot
  - Overview
  - Enumerating Pattern
  - Compute Occurrence
- Conclusions

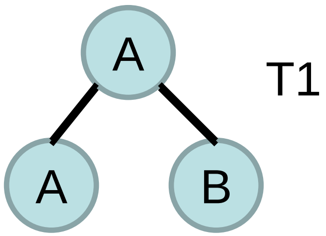
Motivation

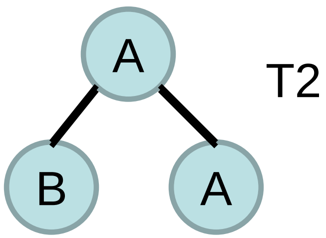
- Graphische Daten sind sehr üblich in vielen Bereichen:
  - Chemie, Biologie, Militär…
- Mit Graphen lassen sich wesentlich komplexere Strukturen darstellen
  - Sind aber auch komplizierter zu handhaben als einfache Mengen
- Graph mining: discovering patterns in large collections of graph or tree structures
- Anwendung:
  - In der Pharmaforschung werden gemeinsame Fragmente einer Menge von Molekülen gesucht

=> ein Algorithumus für gelabelt ungeordnerte Struktur

Motivation

- Wie sieht es aus, wenn Algorithmus z.B. wie Freqt für ungeordnete Bäume angewandt?





$\Rightarrow$ Problem: Es kommt Isomorphie vor

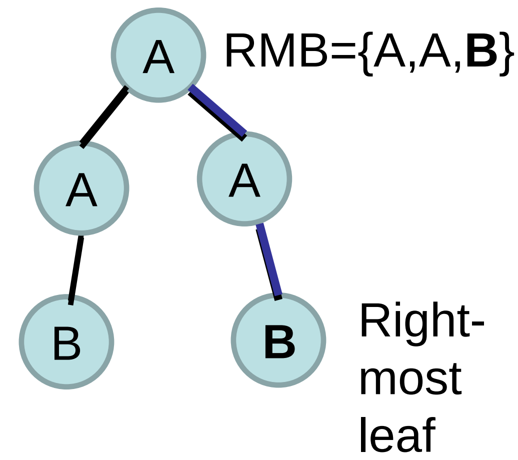
$\Rightarrow$ Lösung: Unot, uFreq, …

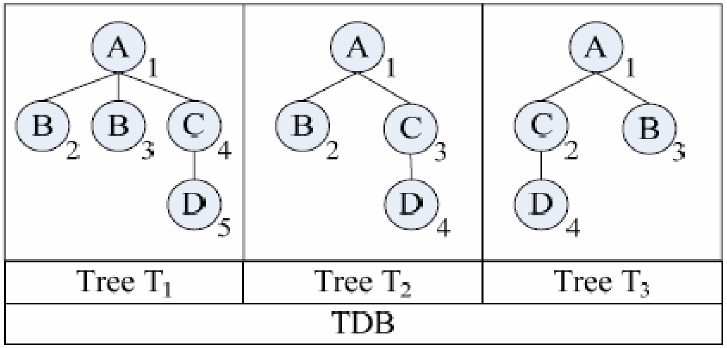
$\Rightarrow$ Wie?

$\Rightarrow$ durch Canonical Form

Definition(1): Basics

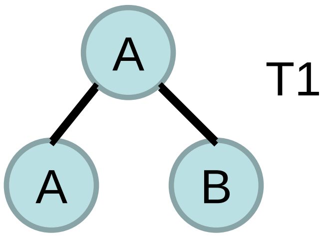
- **Semi-strukturierte Daten**:
  - Hier gelabelt ungeordnet Bäume als ein Modell von semi-strukturierte Daten und Pattern.
- **Labeled unordered Tree**: T=(V,E, r,label)
- **Labeled ordered Tree**: T=(V,E,B, r,label)
  - B: Menge of Geschwester Relation von links zu rechts
  - Z.B: (v1, v2), v1 is the linke Geschwester von v2
- **RMB**(Right most Branch): Pfad von Wurzel zu dem meisten rechten Baumblatt (right most leaf)
- **Unordered database** : is endliche Menge $\mathcal{D}=(D_1 \ldots D_m)$ von (geordnete) Bäume
  - Wobei Di is data tree



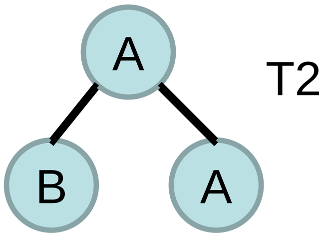


Canonical Form (1)

- gelabelten geordnete Bäume als **Representation** von ungeordneten Bäume
- zwei geordnete Bäum T1, T2 äquivaltent, **T1≡T2**, wenn sie representieren gleiche ungeordnete Bäume=> **Isomorphie**
- **Depth-label sequence C(T)** **C(T)**=((dep(v1),label(v2),…,(dep(vk,label(vk))), wobei jeder Komponent (d,l) als „depth-label Paar“



C(T1) = ((0,A), (1,A), (1,B))



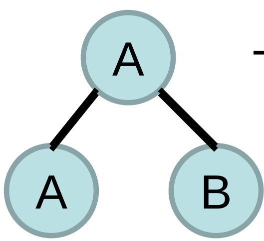
C(T2) = ((0,A), (1,B), (1,A))

Canonical Form(2)

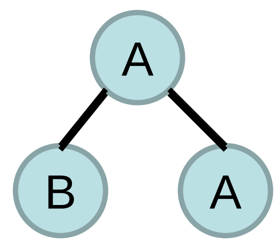
- **Vergleichen** von zwei depth-label Paaren,
- **(d1,l1) > (d2,l2)**
  - wenn d1 > d2 oder
  - d1 = d2 and l1 > l2

Beispiel:
(1,A) > (0,A), wegen d1 = 1 > 0 = d2
(1,A) > (1,B), wegen l1 = A > B = l2
und d1 = d2

- „ **Heavy than**“: **C(T1) $\geqslant$lex C(T2) , wenn i) oder ii)**
- **i)** Vergleich mit depth-label Paar



T1 **$\geqslant$le x** T2

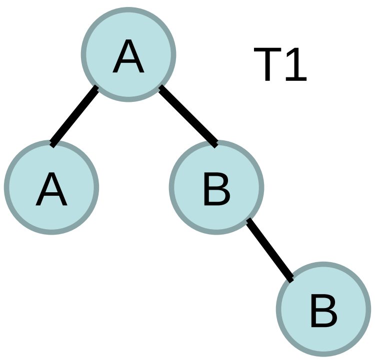


C(T1) = ((0,A), (1,A), (1,B))

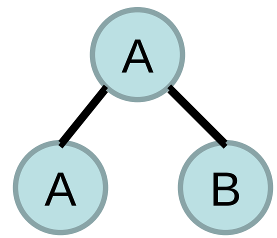
C(T2) = ((0,A), (1,B), (1,A))

Canonical Form(3)

- **ii)** durch Länger der C(T): T2 ist Prefix von T1



T1 $\geqslant_{lex}$ T2



C(T1) = ((0,A), (1,A), (1,B), (2,B))

C(T2) = ((0,A), (1,B), (1,A))

- Ein Baum T ist in Canonical Form, wenn C(T) ist heaviest von allen zu T äquivalenten Bäumen

Canonical Form(4)

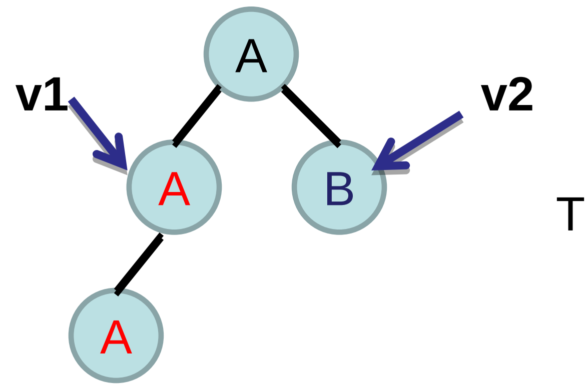
- Jeder kanonische Baum T muss „ left-heavy “ halten.

- **left heavy**: für beliebige Geschwister Knoten v1,v2 impliziert

  $$C(\,T(v1)\,)\ \geqslant\text{lex}\ C(\,T(v2)\,)$$

  => Kriterien für Rightmost Expansion

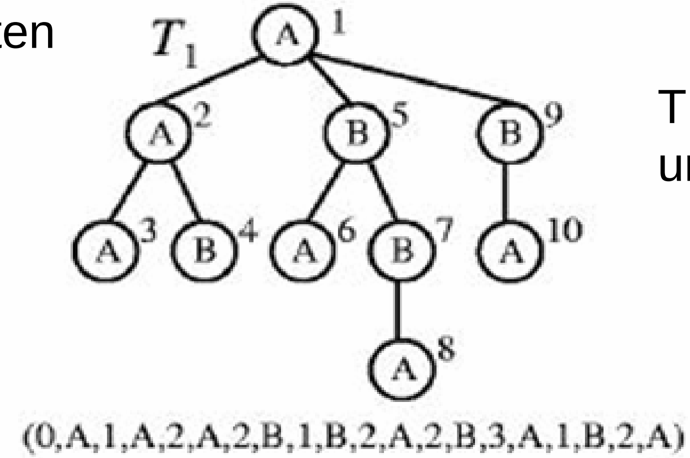
C(T) = ((0,A), (1,A), (1,B))



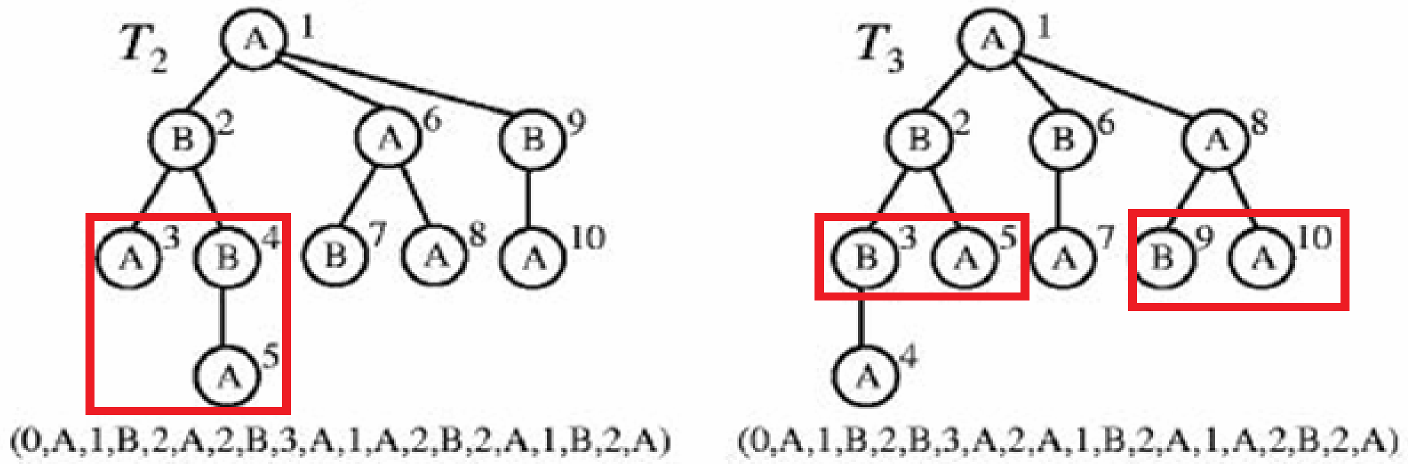
C(T(v1)) = ((0,A), (1A))
$\geqslant$lex
C(T(v2)) = ((0,B))

Beispiel: Left-heavy condition

T1, T2, T3 sind äquivalten
Annahme: A>B>C



T1 ist left heavy
und kanonisch



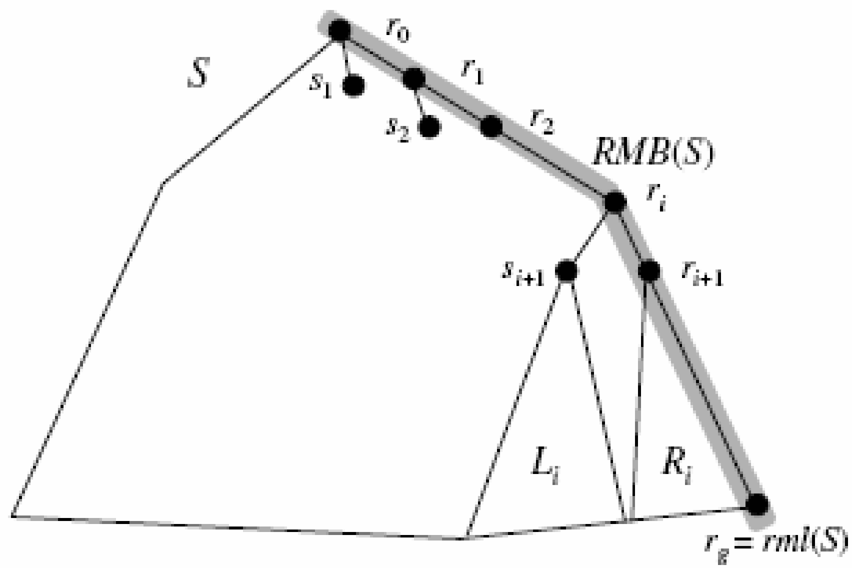
Rightmost expansion

- Wie listen wir alle Subtrees, wenn ein gelabelter geordneter Data Tree vorgegeben?

  => Rightmost expansion

- Reverse Search als allgemeine Idee

- T ist Rightmost expansion of S:
  - durch Hinzufügen eines rightmost Knoten v mit (d,l)
  - T is (d, ℓ)-exentsion of S: S •(d, ℓ)



Unot

- Übersicht:
- Unot: findet alle kanonische Representation für frequent ungeordneten Trees in a vorgegebenem Datenbank $\mathcal{D}$ von Data Trees

- 2 steps:
- i) Enumeration von allen Subtree als Canonical Representation
- => FindAllChildren
- Ii) Computation von ihre Occurrences
- => UpdateOcc

Unot

Algorithm $\mathsf{Unot}(\mathcal{D}, \mathcal{L}, \sigma)$

Input: the database $\mathcal{D} = \{D_1, \ldots, D_m\}$ $(m \geq 0)$ of labeled unordered trees, a set $\mathcal{L}$ of labels, and the minimum frequency threshold $0 \leq \sigma \leq 1$.

Output: the set $\mathcal{F} \subseteq \mathcal{C}$ of all frequent unordered trees of size at most $N_{\mathbf{max}}$.

Method:

1. $\mathcal{F} := \emptyset$; $\alpha := \lceil |\mathcal{D}|\sigma \rceil$; //Initialization
2. For any label $\ell \in \mathcal{L}$, do:
   $T_\ell := (0, \ell)$; /* 1-pattern with copy depth 0 */
   $\boxed{\mathsf{Expand}(T_\ell, \mathcal{O}, 0, \alpha, \mathcal{F});}$
3. Return $\mathcal{F}$; //The set of frequent patterns

Unot

- Input
  - Datenbank von Data Trees
  - Menge von Labels
  - Minimum support threshold σ

- Output
  - Frequent Subtrees die support threshold erreichen

- Initialisierung

Rightmost Expansion: Expand

Procedure Expand$(S, \mathcal{O}, c, \alpha, \mathcal{F})$

Input: A canonical representation $S \in \mathcal{U}$, the embedding occurrences $\mathcal{O} = EO^{\mathcal{D}}(S)$, and the copy-depth c, nonnegative integer α, and the set $\mathcal{F}$ of the frequent patterns.

Method:

– If $(|\mathcal{O}| < \alpha)$ then return;
  Else $\mathcal{F} := \mathcal{F} \cup \{S\}$;

– For each $\langle S \cdot (i, \ell), c_{\text{new}} \rangle \in$ FindAllChildren(S, c), do;

- $T := S \cdot (i, \ell)$;
- $\mathcal{P} :=$ UpdateOcc$(T, \mathcal{O}, (i, \ell))$;
- Expand$(T, \mathcal{P}, c_{\text{new}}, \alpha, \mathcal{F})$;

1) FindAllChildren abrufen um alle Subtrees zu finden

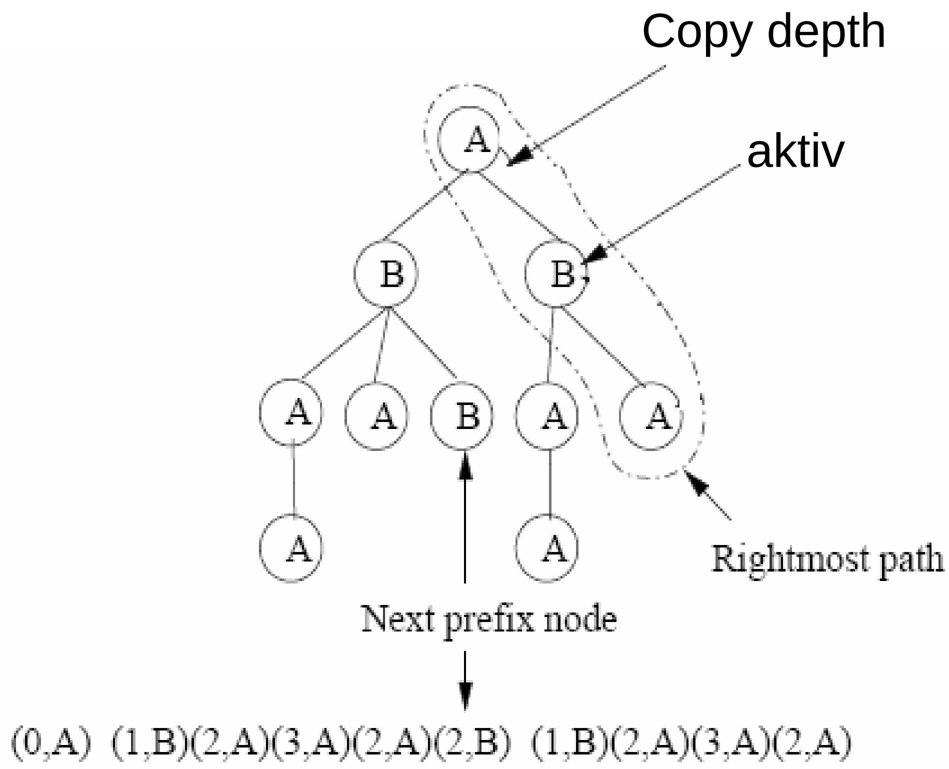
2) UpdateOcc abrufen für jeden Subtree, um jeweilige Occurence zu rechnen

3) test, ob der gefundene Subtree frequent ist

4) neue frequente Subtree in Output Mengen hinzufügen

Copy Depth

- Interne Knoten am RMB ri ist **aktiv**, wenn C(Ri) = C(Li)
- Copy Depth: höchste aktive Knoten
- Rml als spezielle Copy Depth wenn es keine andere gibt.



Enumeration: FindAllchildren

Procedure FindAllChildren(T, c) :

Method : Return the set $Succ$ of all pairs $\langle S, c\rangle$, where S is the canonical child tree of T and c is its copy depth generated by the following cases:

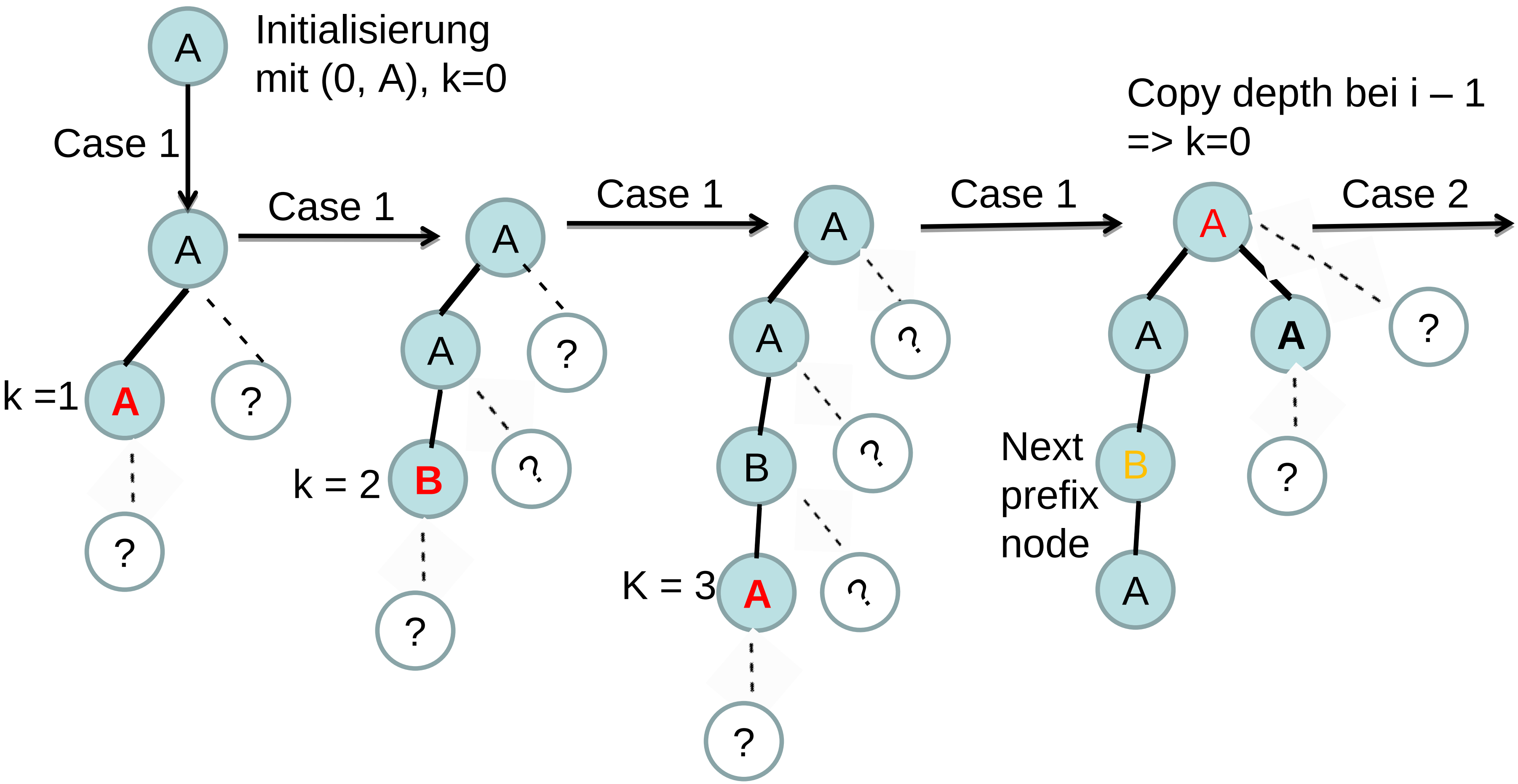
Case I : If $C(L_k) = C(R_k)$ for the copy depth k:

- The canonical child trees of T are $T\cdot(1, \ell_1), \ldots, T\cdot(k+1, \ell_{k+1})$, where $label(r_i) \geq \ell_i$ for every $i = 1, \ldots, k+1$. The trees $T\cdot(k+2, \ell_{k+2}), \ldots, T\cdot(g+1, \ell_{g+1})$ are not canonical.
- The copy depth of $T\cdot(i, \ell_i)$ is $i-1$ if $label(r_i) = \ell_i$ and i otherwise for every $i = 1, \ldots, k+1$.

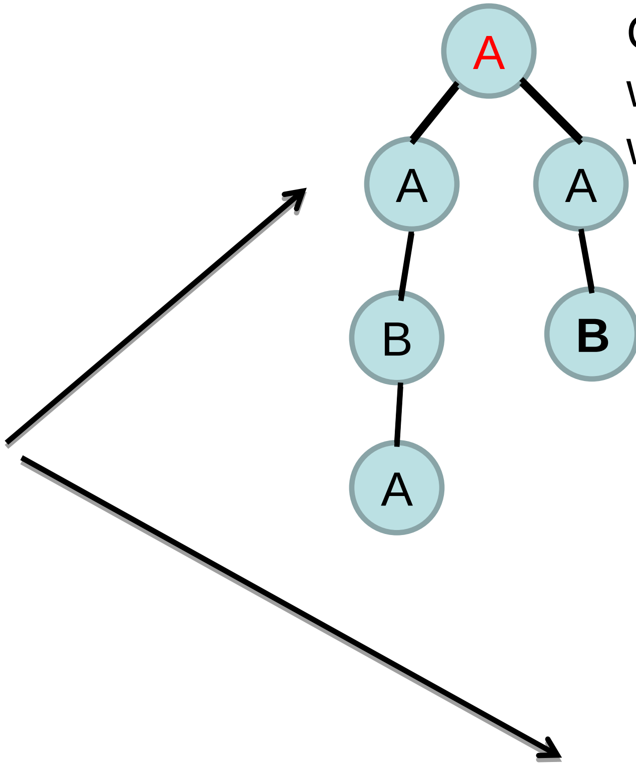
Case II : If $C(L_k) \neq C(R_k)$ for the copy depth k:

- Let $m = |C(R_k)|+1$ and $w = (d, \ell)$ be the m-th component of $C(L_k)$ (the next position to be copied). The canonical child trees of T are $T\cdot(1, \ell_1), \ldots, T\cdot(d, \ell_d)$, where $label(r_i) \geq \ell_i$ for every $i = 1, \ldots, d-1$ and $\ell \geq \ell_d$ holds.
- The copy depth of $T\cdot(i, \ell_i)$ is $i-1$ if $label(r_i) = \ell_i$ and i otherwise for every $i = 1, \ldots, d-1$. The copy depth of $T\cdot(d, \ell_d)$ is k if $w = v$ and d otherwise.

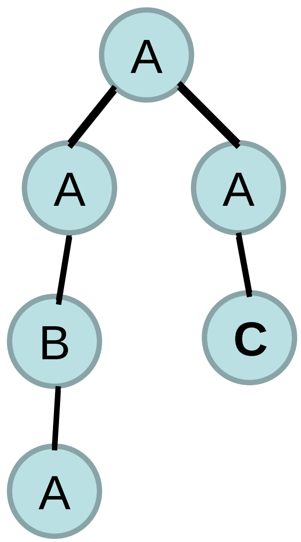
Beispiel



A
Initialisierung
mit (0, A), k=0
Case 1
A
k =1
A
?
?
Case 1
A
A
?
k = 2
B
?
?
Case 1
A
A
?
B
?
K = 3
A
?
?
Copy depth bei i – 1
=> k=0
Case 1
A
A
A
?
Next
prefix
node
B
?
A
Case 2

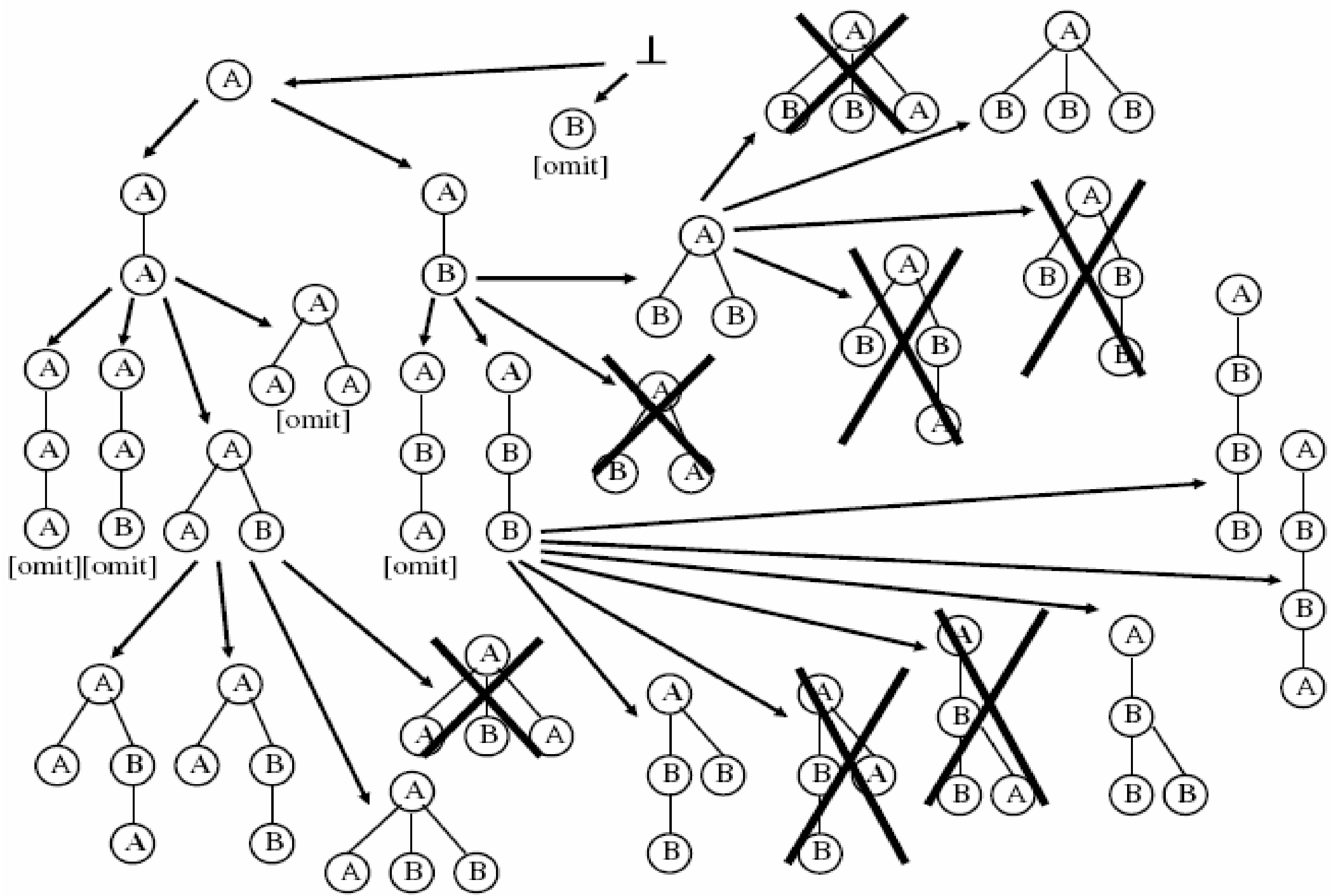


Copy depth bleibt bei (0, A), wenn rechte Subtree kopiert weiter



Kopiert nicht mehr weiter, copy depth ändert

Beispiel:

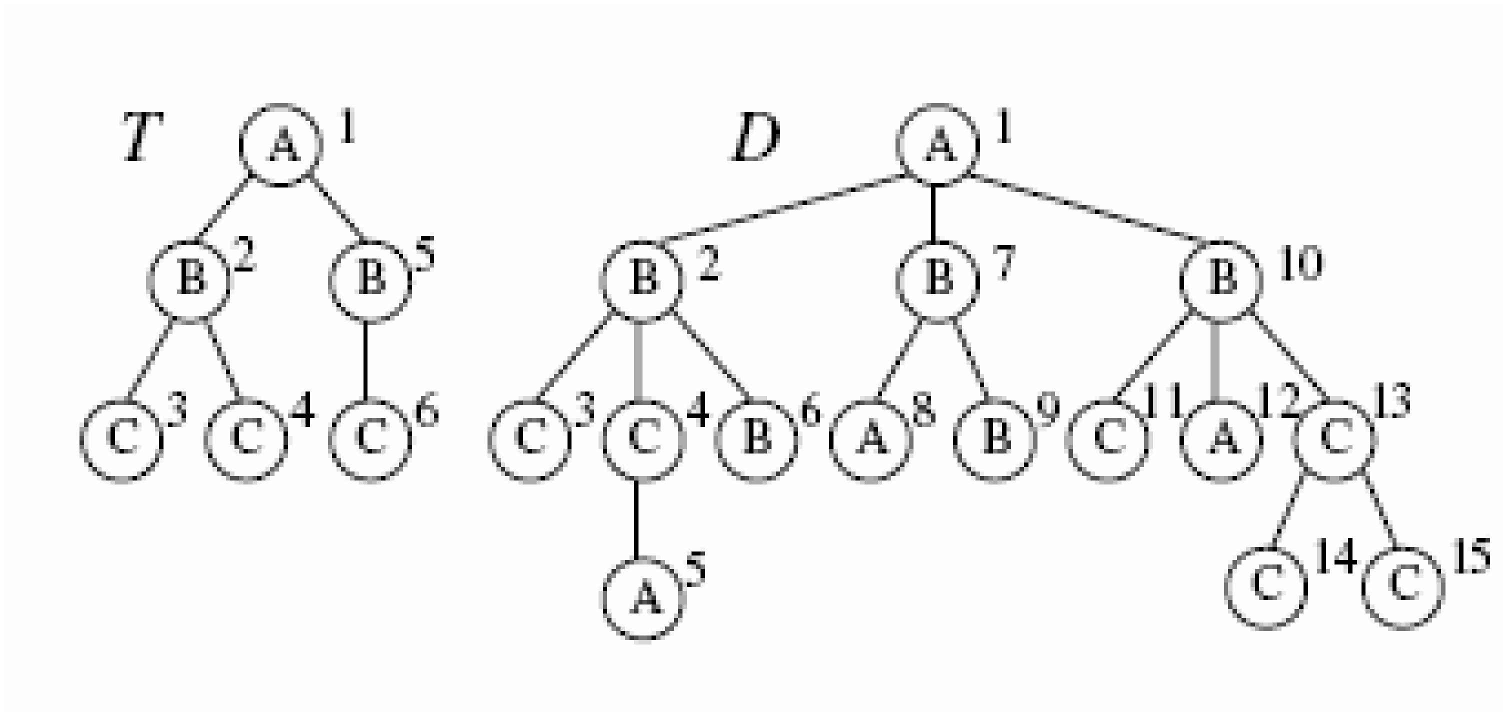


Definition(2): Occurrences

- Occurrence: T occurs in D
  - Es existert mapping ϕ : V_D --> V_T
  - die Eltern Relation beibehalten
  - Und Labels beibehalten
- Variant: total, embedding, root, document.

- Total occurrence: is the k Tupple $Toc(\phi) = < \phi(1), \ldots \phi(k) > \in (V_D)^k$
- Embedding occurrence: is the set $Eoc(\phi) = \{ \phi(1), \ldots \phi(k) \}$ gehört zu V_D
- Root occurrence: $Roc(\phi) = \phi(1) \in V_D$

Beispiel: Occurrences



Updateocc (1)

Algorithm UpdateOcc$(T, \mathcal{O}, d, \ell)$

Input: the rightmost expansion of a pattern S, the embedding occurrence list $\mathcal{O} = EO^{\mathcal{D}}(S)$, the depth $d \geq 1$ and a label $\ell \in \mathcal{L}$ of the rightmost leaf of T.

Output: the new list $\mathcal{P} = EO^{\mathcal{D}}(T)$.

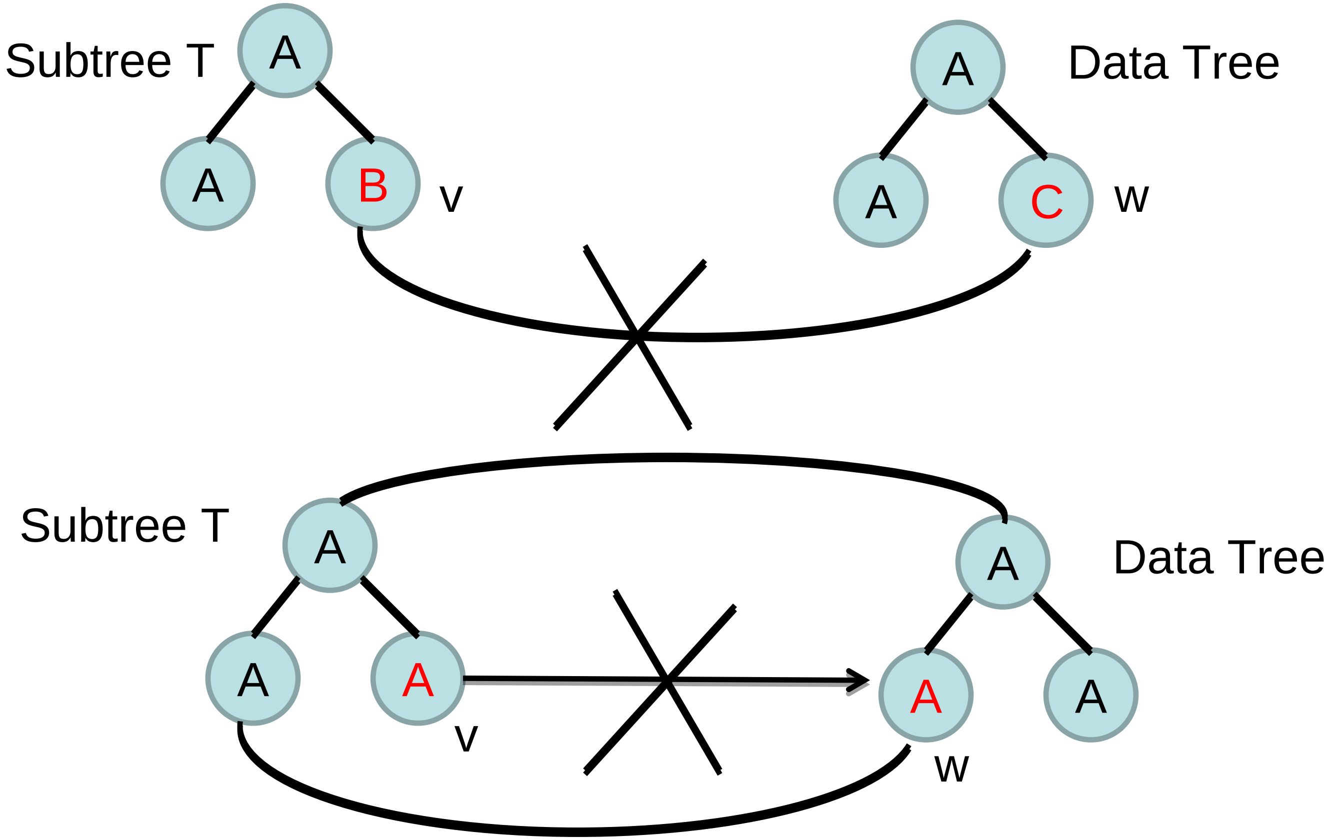
Method:

- $\mathcal{P} := \emptyset$;
- For each $\varphi \in \mathcal{O}$, do:
  - + $x := \varphi(r_{d-1})$; /* the image of the parent of the new node $r_d = (d, \ell)$ */
  - + For each child y of x do: Bedingung 3)
    - – If $label_D(y) = \ell$ and $y \notin E(\varphi)$ then Bedingung 1), 2)
      $\xi := \varphi \cdot y$ and $flag := true$;
    - – Else, skip the rest and continue the for-loop;
    - – For each $i = 1, \ldots, d-1$, do:
      If $C(L_i) = C(R_i)$ but $\xi(left_i) \neq \xi(right_i)$ then Bedingung 4)
      $flag := false$, and then break the inner for-loop;
    - – If $flag = true$ then $\mathcal{P} = \mathcal{P} \cup \{\xi\}$;
- Return $\mathcal{P}$;

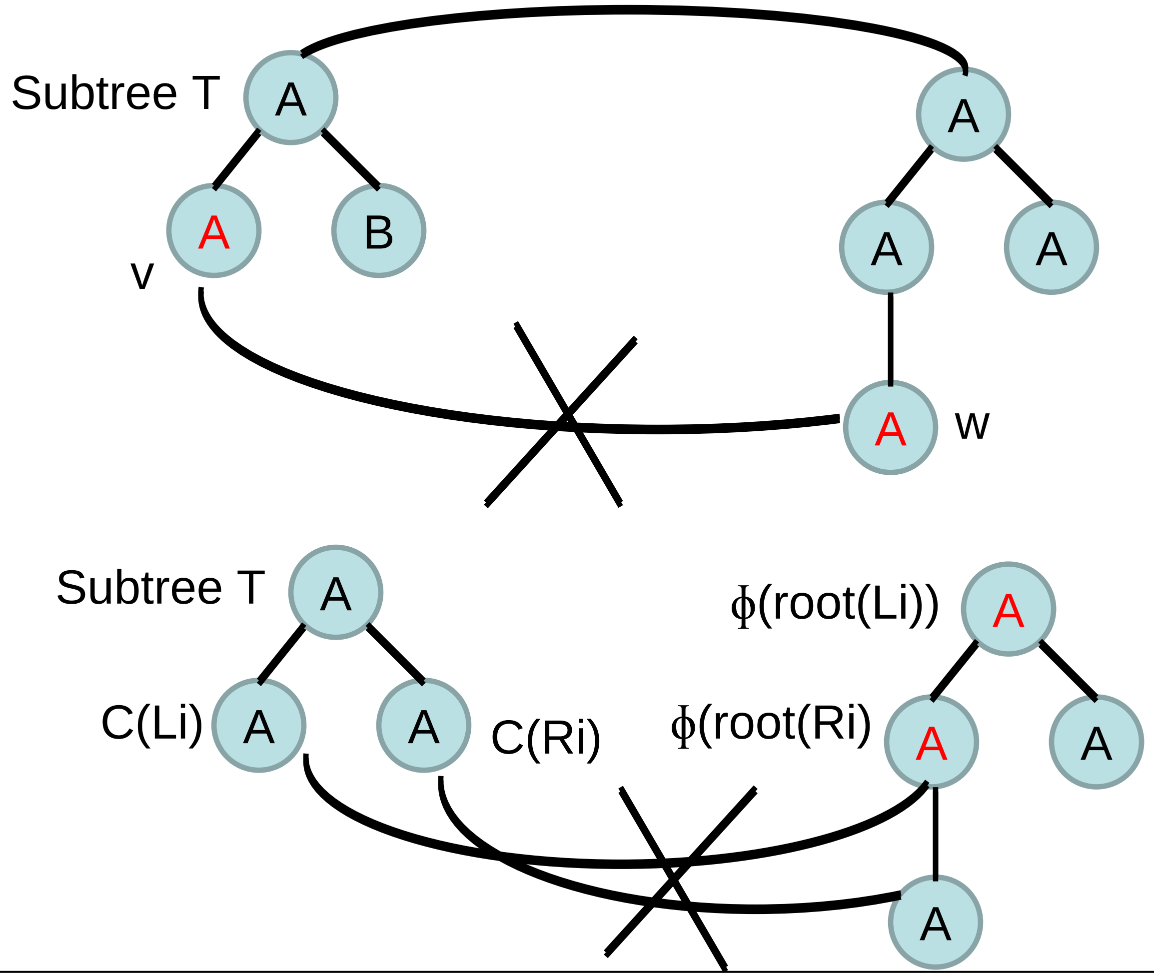
Updateocc (2)

- Ein Mapping $\phi = \Phi \bullet w$ ist ein kanonische total Occurrence von Subtree T in $\mathcal{D}$, wobei Φ ist Occurrence Liste der Vorfahren von Subtree T, wenn es folgende Bedingungen erfüllt:
- 1) $Label_T(v) = Label_{\mathcal{D}}(w)$
- 2) w trat noch nicht in der Occurrence Liste der Vorfahren von Subtree T
- 3) wenn v ist ein Kinder von r_i, muss w auch ein Kinder von $\Phi\ (r_i)$
- 4) wenn C(Li) = R(i), impliziert $\phi(root(Li)) = \phi(root(Ri))$
  - Nachbildung von Li sowie Ri in $\mathcal{D}$ haben auch gleichen Elternknoten.

Beispiel



Beispiel



Subtree T
A
A
B
v
A
A
A
A
w
Subtree T
A
C(Li)
A
A
C(Ri)
ϕ(root(Li))
A
ϕ(root(Ri))
A
A
A

Zeitkomlexität

- Konstante Zeitkomplexität O(1) für die Enumeration jedes Subtree
- Enumeration für jedes frequent pattern T in $O(kb^2m)$, wobei k und b jeweils Größe und Branching Fator von T, sowie m ist total occurrences of T in Data Trees

Fazit

- Depth-first Traverse
- Enumeration based on Canonical Representation => Isomophie Problem vermeiden (effizienter, schneller)
- Nutzen von Rightmost Expansion
- Nutzen von Occurrence Liste zum Rechnen von Menge von frequent canonical Subtrees

Literature

- [1] Asai, T., Abe, K., Kawasoe, S., Arimura,H.: Efficient Substructure Discovery from Large Semi-Structured Data, 2002.

- [2] K. Abe, S. Kawasoe, T. Asai, H. Arimura, and S. Arikawa. Optimized Substructure Discovery for Semi-structured Data,*2002.*

- [3] Asai, T., Abe, K., Kawasoe, S., Arimura,H.: Efficient Substructure Discovery from Large Semi-structured Data

- [4] S. Nakano, T. Uno, Efficient Generation of Rooted Trees, *NII Techn ical Report, 2002*

Thank you ! ☺

Comparison uFreq and Unot

- same:
- Rightmost expansion
- Use occrence list

- Difference:
- Freq: rekord occrence list of every pattern (more memory cost, more complex occurrence list)
- Unot: rekord occrence list of every <u>canonical pattern</u>

Basic definition

- Pre-order-string: same to depth label sequece