Efficiently mining Frequent Trees in a Forest: Algorithms and Applications

Seminar in maschinellem Lernen – Prof. Fürnkranz – Christoph Stock

Überblick:

- Motivation

- subtrees und Wälder

- TreeMiner für geordnete Bäume
    - Listen erstellen
    - Baumkandidaten erweitern
    - Listen verknüpfen
    - Optimierungen
    - TreeMinerD

- PatternMatcher
- Messergebnisse

- SLEUTH für ungeordnete Bäume
    - Ordnung auf Bäumen
    - Erzeugen von Baumkandidaten
    - Unterschiede im Verknüpfen der Listen

- Fazit

Efficiently mining Frequent Trees in a Forest: Algorithms and Applications

Seminar in maschinellem Lernen – Prof. Fürnkranz – Christoph Stock

Motivation:

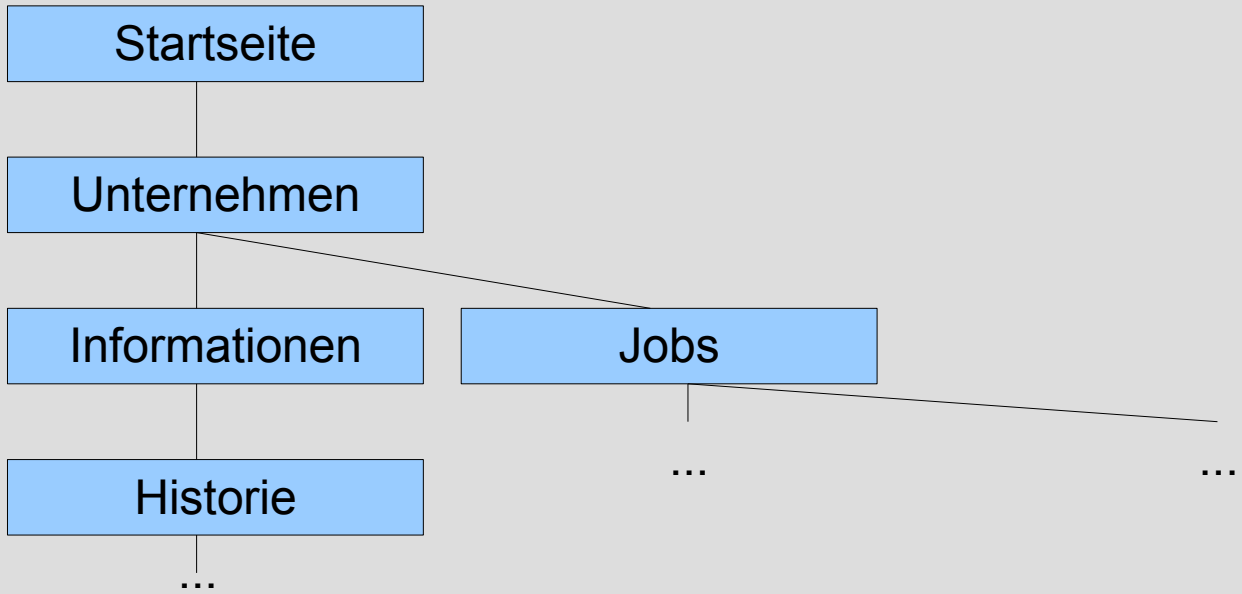
- Datensatz besteht aus Logdateien über eine Webseite, jeder Besucher wird als Baum gespeichert

- Wir wollen häufiges Verhalten auf der Seite feststellen um die Navigation zu verbessern

- Wir suchen häufige Subtrees in den Daten (Forest)

Beispiel:

ein Besuch einer Webseite



Efficiently mining Frequent Trees in a Forest: Algorithms and Applications

Seminar in maschinellem Lernen – Prof. Fürnkranz – Christoph Stock

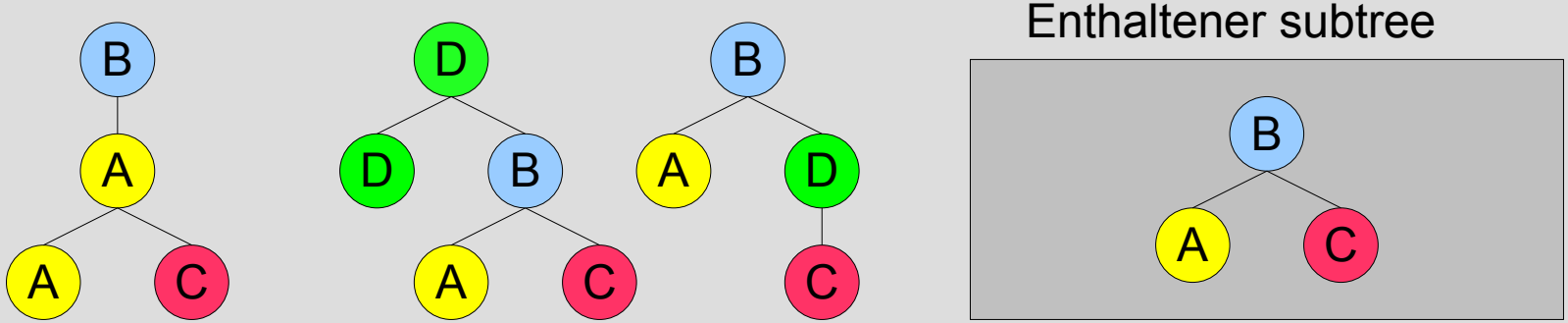
Problem:

Gegeben: ein Wald aus Bäumen

Gesucht: häufige subtrees in den Bäumen

Einschränkung: zunächst betrachten wir geordnete Bäume

Was ist ein subtree?

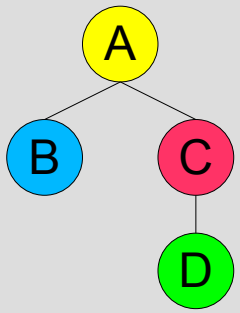


Efficiently mining Frequent Trees in a Forest: Algorithms and Applications

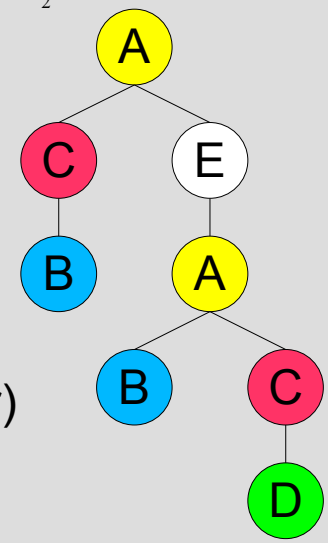
Seminar in maschinellem Lernen – Prof. Fürnkranz – Christoph Stock

Datenbank D:

T_0

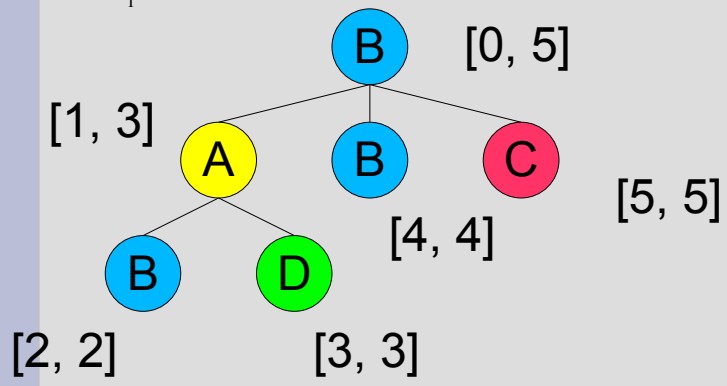


T_2



Scope:
(ID in DFS, höchster Nachfolger)

T_1



D_{HORZ}:

T_0: A B $ C D $ $

T_1: B A B $ D $ $ B $ C $

T_2: A C B $ $ E A B $ C D $ $ $ $

In vertikaler Ordnung schreiben wir zu jedem Label TreeID und Scope auf:

D_{VERT}:

A:

$0,(0,3);1,(1,3);$

$2,(0,0);2,(4,7);$

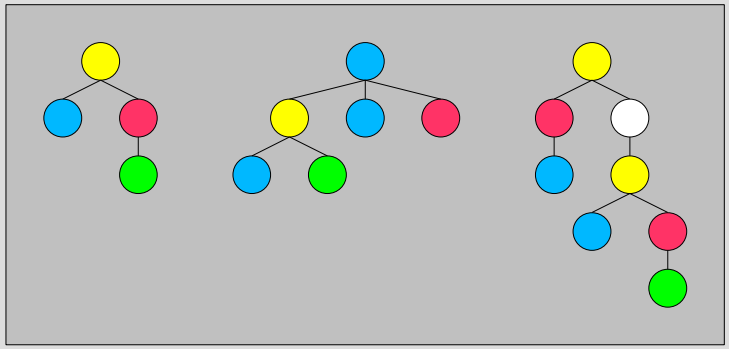
B:

...

Efficiently mining Frequent Trees in a Forest: Algorithms and Applications

Seminar in maschinellem Lernen – Prof. Fürnkranz – Christoph Stock

TreeMiner: *Erstellen von Listen mit frequenten Label*:



- nur Label die in minsup Bäumen vorkommen, werden aufgenommen
- im Beispiel minsup = 100%

Prefix = {}

Elements: (1, -1), (2, -1), (3,-1), (4,-1)

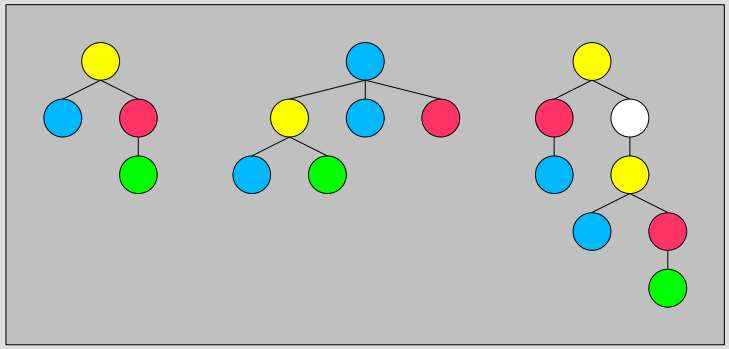
A	B	C	D
0, [0, 3]	0, [1, 1]	0, [2, 3]	0, [3, 3]
1, [1, 3]	1, [0, 5]	1, [5, 5]	1, [3, 3]
2, [0, 7]	1, [2, 2]	2, [1, 2]	2, [7, 7]
2, [4, 7]	1, [4, 4]	2, [6, 7]	
	2, [2, 2]		
	2, [5, 5]		

Elements sind Ersteller (x, i), die an dem Knoten mit der DFS-ID i einen Knoten mit Label x anhängen.
Wenn i=-1 wird ein neuer Baum angelegt

Efficiently mining Frequent Trees in a Forest: Algorithms and Applications

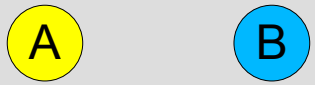
Seminar in maschinellem Lernen – Prof. Fürnkranz – Christoph Stock

TreeMiner: *Verknüpfen von frequenten subtrees*:

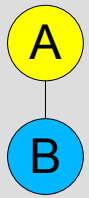


zunächst werden die Elemente verknüpft, um die Erzeuger für den nächsten Schritt zu erstellen

$(x,i)\otimes(y,j)$
$1.\ i=j$
(y,j) Cousin-Extension
(y,n) Sohn-Extension, n ist DFS-ID von x
$2.\ i>j$
(y,j) Cousin-Extension

$Beispiel:(A,-1)\otimes(B,-1)$ 

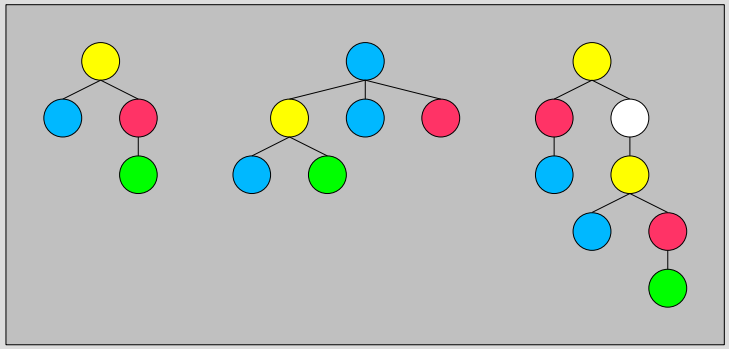
daraus wird erzeugt
$(B,-1)$ ist ein bereits erstellter subtree und wird verworfen
$(B,0)$ wird der Liste der 2-subtrees hinzugefügt und die Scope-Listen der Ausgangsbäume werden verknüpft



Efficiently mining Frequent Trees in a Forest: Algorithms and Applications

Seminar in maschinellem Lernen – Prof. Fürnkranz – Christoph Stock

TreeMiner: *Verknüpfen von frequenten subtrees*:



- Verknüpfen der Scope-Listen der einzelnen Elemente
- Häufigkeit des subtree direkt ablesbar aus neuer Scope-Liste

Wir brauchen eine Ordnung auf Scopes

$s_x = (l_x, u_x); s_y = (l_y, u_y)$

$s_x < s_y$ iff $u_x < l_y$

$s_x \supset s_y$ iff $l_x \leq l_y$ und $u_x \geq u_y$

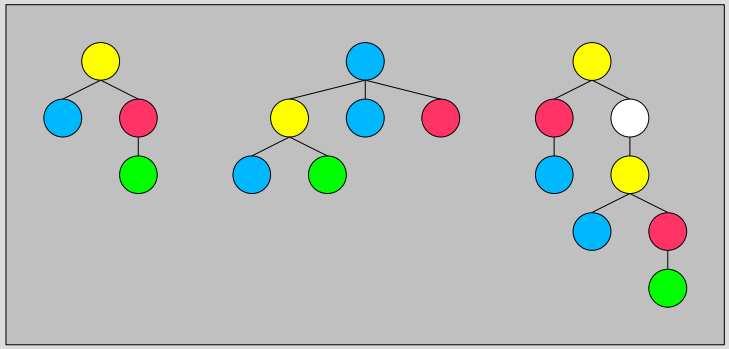
1. Relation: Y ist Cousin zu X und liegt in einem Ast weiter rechts

2. Relation: Y ist Nachfahre von X

Efficiently mining Frequent Trees in a Forest: Algorithms and Applications

Seminar in maschinellem Lernen – Prof. Fürnkranz – Christoph Stock

TreeMiner: *Verknüpfen von frequenten subtrees:*



Wir vergleichen die Tripel (t_x, m_x, s_x) und (t_y, m_y, s_y)

t ist TreeID, m ist eine MatchID, s ist der Scope

$$t_x = t_y$$

$$m_x = m_y$$

Wenn Sohn-Extension

$$s_x \supset s_y$$

Wenn Cousin-Extension

$$s_x < s_y$$

Wenn alle drei Bedingungen gelten, fügen wir der neuen ScopeList

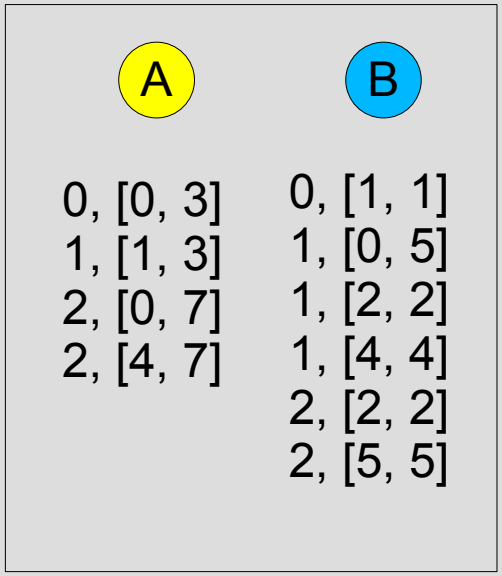
$$(t_y, m_y \cup l_x, s_y)$$

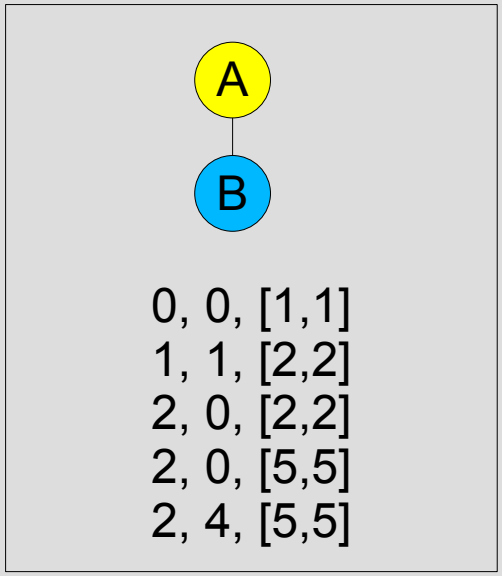
hinzu

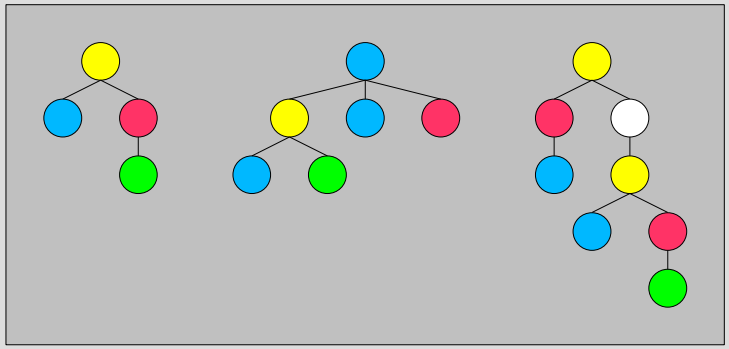
Efficiently mining Frequent Trees in a Forest: Algorithms and Applications

Seminar in maschinellem Lernen – Prof. Fürnkranz – Christoph Stock

TreeMiner: *Verknüpfen von frequenten subtrees:*







$$(0, \emptyset, [0,3]) \otimes (0, \emptyset, [1,1])$$

ist in einer Sohn-Extension entstanden, also

$$0 = 0$$

$$\emptyset = \emptyset$$

$$[0,3] \supset [1,1] \text{ weil } 0 \leq 1 \text{ und } 3 \geq 1$$

Daraus entsteht das Element der neuen Scopeliste

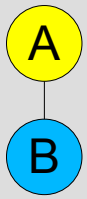
$$(0,0,[1,1])$$

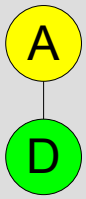
Efficiently mining Frequent Trees in a Forest: Algorithms and Applications

Seminar in maschinellem Lernen – Prof. Fürnkranz – Christoph Stock

TreeMiner: *Verknüpfen von frequenten subtrees:*

Prefix = A
Elemente = (A,0), (D,0)

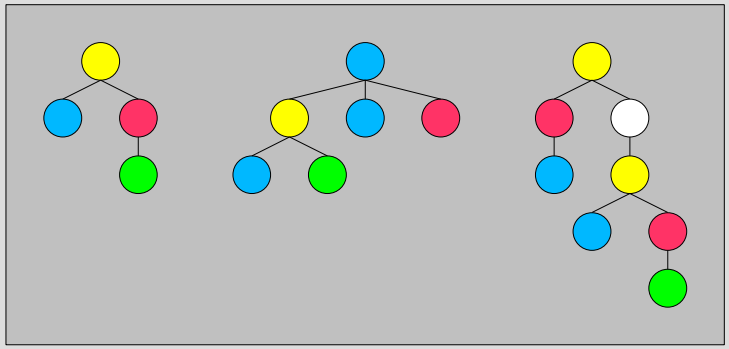




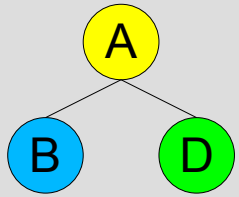
0, 0, [1,1]
1, 1, [2,2]
2, 0, [2,2]
2, 0, [5,5]
2, 4, [5,5]

0,0,[3,3]
1,1,[3,3]
2,0,[7,7]
2,4,[7,7]

Alle anderen Kombinationen ergeben infrequente subtrees



Prefix = AB
Elemente = (D,0)



0,01,[3,3]
1,12,[3,3]
2,02,[7,7]
2,05,[7,7]
2,45,[7,7]

Efficiently mining Frequent Trees in a Forest: Algorithms and Applications

Seminar in maschinellem Lernen – Prof. Fürnkranz – Christoph Stock

Optimierungen für TreeMiner

Speicherersparnis

- wir können das Speichern der Matchlabel entfallen lassen, wenn ein subtree nur einmal in einem Baum vorkommt.

- Tests zeigen, dass dies den Speicherverbrauch drastisch reduzieren kann

Kandidaten-Pruning

- Wir können das Erstellen von Kandidaten und Verknüpfen von Scope-Lists im voraus verhindern, wenn wir wissen, dass der zu erstellende subtree bereits gezählte infrequente Elemente enthält. (In den Beispielen schon gezeigt)

Efficiently mining Frequent Trees in a Forest: Algorithms and Applications

Seminar in maschinellem Lernen – Prof. Fürnkranz – Christoph Stock

Variation des Algorithmus: **TreeMinerD**

- im Gegensatz zu TreeMiner findet TreeMinerD immer nur Vorkommen des subtree in jedem Baum

- der Algorithmus bleibt gleich, es wird eine andere Repräsentation der Scope-Liste gewählt: (t, s)

$$s = s_{1,} s_{2,} .. s_n$$

s sind die Knoten auf dem weitest rechts liegenden Pfad

Efficiently mining Frequent Trees in a Forest: Algorithms and Applications

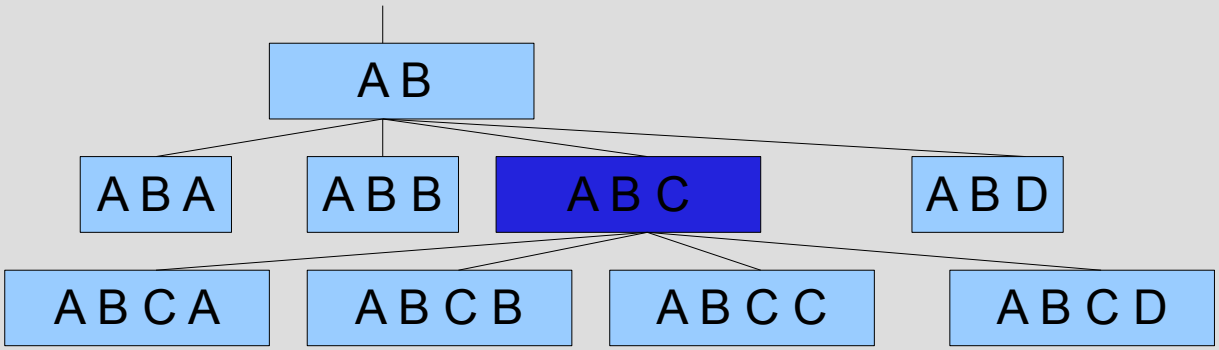
Seminar in maschinellem Lernen – Prof. Fürnkranz – Christoph Stock

Ein Algorithmus zum Vergleich: **PatternMatcher**

- BFS statt DFS, dadurch können potentielle subtree-Kandidaten frühzeitig verworfen werden

- der Algorithmus arbeitet auf dem String, also der horizontalen Ordnung

- PrefixDataStructure: ein Knoten enthält alle Baumvariationen einer Knotenreihenfolge (Beispiel: Knoten (A B C) enthält „A B C $ $“, „A B $ C $“)

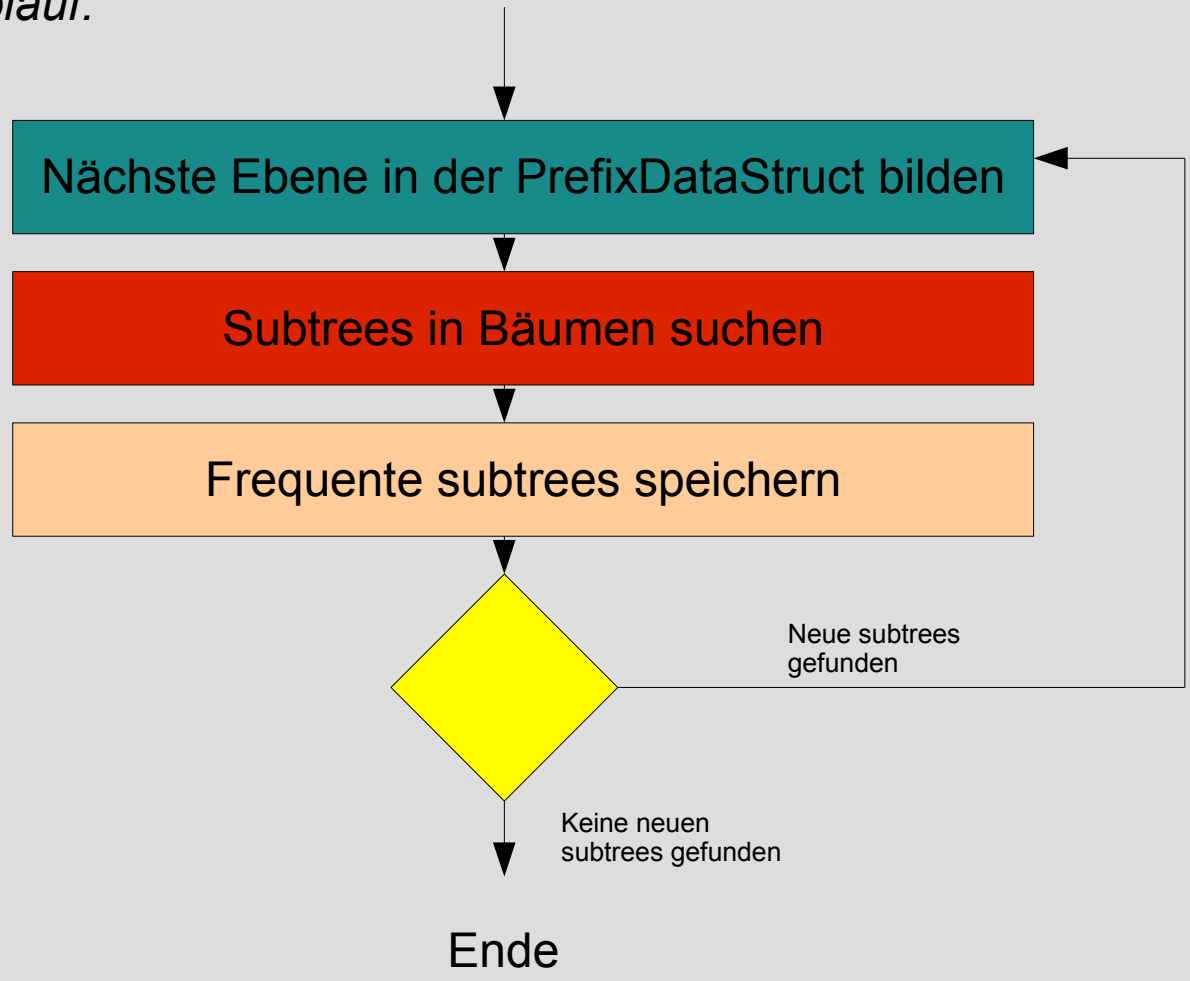


Efficiently mining Frequent Trees in a Forest: Algorithms and Applications

Seminar in maschinellem Lernen – Prof. Fürnkranz – Christoph Stock

Ein Algorithmus zum Vergleich: **PatternMatcher**

- Ablauf:



Efficiently mining Frequent Trees in a Forest: Algorithms and Applications

Seminar in maschinellem Lernen – Prof. Fürnkranz – Christoph Stock

Messergebnisse:

- Trotz des Vorteils der BF-Suche kann der PatternMatcher auf keinem der Testsets die beiden TreeMiner-Varianten schlagen.

- TreeMiner und TreeMinerD verhalten sich in der Laufzeit in etwa gleich, bei sehr kleinen minimalem Support wird TreeMinerD besser

- Laufzeiten bei ca. 60.000 Bäumen (Homepage-Log mit 13.000 Webpages) bei 7 sek für TreeMiner(D), für eine Million Bäumen (generisch, 100 Label) ca. 30 sek

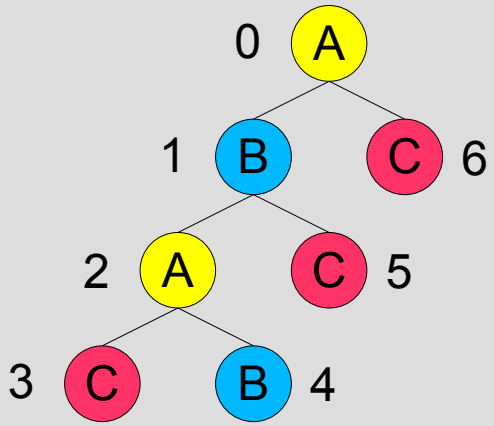
- Pruning verbessert die Laufzeit um das zwei- bis vierfache

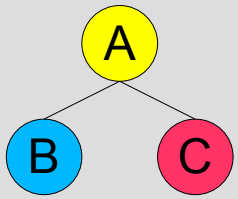
Efficiently mining Frequent Trees in a Forest: Algorithms and Applications

Seminar in maschinellem Lernen – Prof. Fürnkranz – Christoph Stock

SLEUTH

- wir suchen nun in ungeordneten Bäumen embedded subtrees





Embedded, unordered sub-tree
match labels:
{016, 045, 046, 043, 243}

d.h. Die Kinder im subtree müssen nur irgendwann Nachfolger im Baum sein, also unabhängig von Reihenfolge oder der Abhängigkeit untereinander

Efficiently mining Frequent Trees in a Forest: Algorithms and Applications

Seminar in maschinellem Lernen – Prof. Fürnkranz – Christoph Stock

SLEUTH

- Wir definieren eine Ordnung auf Bäume (es existiert eine Ordnung der Label)

r_x und r_y sind die root-Knoten der Bäume X und Y

$c_i^{(r_x)}$ und $c_i^{(r_y)}$ ist eine geordnete Liste aller Kinder

$T(c_i^{(r_x)})$ ist der Subtree X ab Knoten $c_i^{(r_x)}$

$X \leq Y$ iff

1.) $l(r_x) < l(r_y)$ oder

2.) $l(r_x) = l(r_y)$ und

a) $n \leq m$ und $T(c_i^{(r_x)}) = T(c_i^{(r_y)})$ für alle $i \leq n$ oder

b) es gibt ein $j \in [1, min(n, m)]$, so dass $T(c_i^{(r_x)}) = T(c_i^{(r_y)})$ für alle $i < j$ und $T(c_j^{(r_x)}) < T(c_j^{(r_y)})$

Fall a: Die Bäume sind gleich, bzw Y ist ein Prefix von X
Fall b: Es existiert ein Label in X, dass echt kleiner ist

Efficiently mining Frequent Trees in a Forest: Algorithms and Applications

Seminar in maschinellem Lernen – Prof. Fürnkranz – Christoph Stock

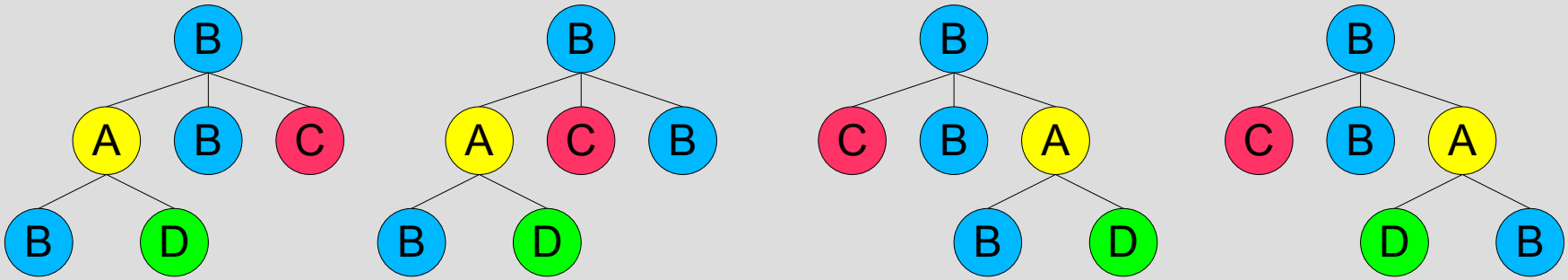
SLEUTH

- Bäume, die sich nur durch die Reihenfolge der Knoten unterscheiden nennen wir automorph.

- der kanonische Vertreter der automorphen Gruppe ist der nach der definierten Ordnung kleinstmögliche Baum der Gruppe

- die Ordnung lässt sich auch auf die String-Repräsentation übertragen

Automorphe Bäume:



Kanonischer Vertreter

Efficiently mining Frequent Trees in a Forest: Algorithms and Applications

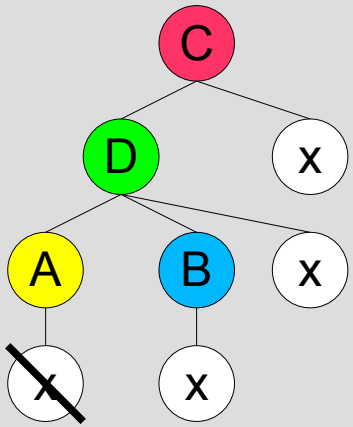
Seminar in maschinellem Lernen – Prof. Fürnkranz – Christoph Stock

SLEUTH

Das Erzeugen der Kandidaten bei SLEUTH ist ein TradeOff zwischen kanonischen und wahrscheinlich frequenten Bäumen.

Grundlage: Präfix Extension

- zu einem gegeben Präfix ins String-Notation werden alle vorhanden Label hinzugefügt



Präfix: C D A $ B

Extension: x, $ x, $ $ x

Anhängen an A nicht möglich, weil dann der Präfix verändert wird

Efficiently mining Frequent Trees in a Forest: Algorithms and Applications

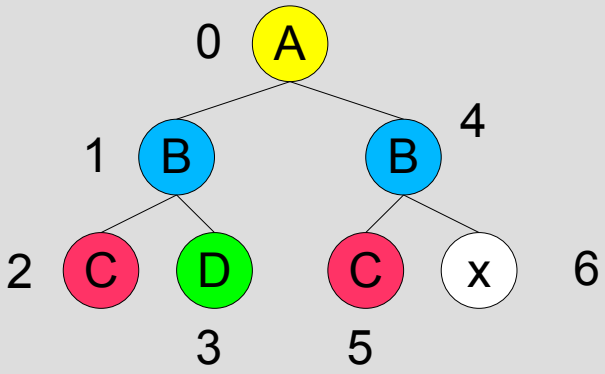
Seminar in maschinellem Lernen – Prof. Fürnkranz – Christoph Stock

SLEUTH

Möglichkeit 1: *Kanonische Extension*

zu jeder Extension wird geprüft, ob sie wieder einen kanonischen Baum erzeugt

Beispiel:



Anfügen von Knoten 6

- Durch Überprüfen der Kinder von 4 wird verlangt, dass x >= C

- rekursiver Durchlauf ergibt weiter, dass x=D, da sonst wegen 3 die kanonische Form verletzt wäre

Vorteil: Jeder subtree-Kandidat wird genau einmal erzeugt

Nachteil: Aufwändige Berechnung, infrequente Subbäume werden immer wieder gebildet

Efficiently mining Frequent Trees in a Forest: Algorithms and Applications

Seminar in maschinellem Lernen – Prof. Fürnkranz – Christoph Stock

SLEUTH

Möglichkeit 2: *Equivalenzklassen-Erweiterung*

- Die Bäume werden wie bei TreeMiner über die Elemente (x, i) erweitert.

Nachteil: dies führt zu vielen Vertretern die einer Automorphismus-Gruppe angehören

Vorteil: Es werden aber Bäume nur mit frequenten Teilbäumen erweitert

Verbesserung: Nach jedem Erweiterungsschritt werden alle Vertreter einer Automorphismus-Gruppe entfernt, die sich nicht in der kanonischen Form befinden

Tests zeigen, dass die Equivalenzklassen-Erweiterung im Vergleich zur kanonischen Erweiterung bis zu fünfmal so schnell ist

Efficiently mining Frequent Trees in a Forest: Algorithms and Applications

Seminar in maschinellem Lernen – Prof. Fürnkranz – Christoph Stock

SLEUTH

Das bilden der Scope-Listen und Verknüpfen nach Sohn-Erweiterungen ist identisch zu TreeMiner

Cousin-Erweiterung:

$$t_x = t_y$$
$$m_x = m_y$$
$$s_x < s_y \text{ oder } s_x > s_y$$

prüft sowohl größer als auch kleiner, da die Kinder in den Bäumen ungeordnet sind

Efficiently mining Frequent Trees in a Forest: Algorithms and Applications

Seminar in maschinellem Lernen – Prof. Fürnkranz – Christoph Stock

Fazit:

- TreeMiner für geordnete Bäume ist deutlich schneller als der Apriori-ähnliche Ansatz PatternMatcher

- SLEUTH für ungeordnete Bäume arbeite mit den redundanten Equivalenzklassen-Erweiterungen deutlich schneller als mit den kanonischen Erweiterungen

- entscheidend ist, dass die Erweiterung immer mit großer Wahrscheinlichkeit zu frequenten subtrees führt

Ausblick:

- Benutzer-Constraints um die Baum-Erweiterung den gewünschten Ergebnissen anzupassen

Efficiently mining Frequent Trees in a Forest: Algorithms and Applications

Seminar in maschinellem Lernen – Prof. Fürnkranz – Christoph Stock

Vielen Dank für die Aufmerksamkeit