



Vorlesung Wissensentdeckung in Datenbanken

Tree Projection – LTree

Katharina Morik, Uwe Ligges

Informatik LS 8
Computergestützte Statistik
Technische Universität Dortmund

27.4.2010



Gliederung

- 1 Einführung
- 2 LTrees
- 3 LTrees zum frequent set mining
- 4 Vor- und Nachteile



Tree Projection – LTree

- Man kann statt FPtrees auch LTrees verwenden.
- Implementierungsdetails von frequent set mining FIMI workshops (Goethals, Zaki)
- Aspekte, die bedacht werden:
 - Speicherbedarf
 - Passt in Hauptspeicher?
 - Passt in Cache?
 - I/O Zugriffe auf Datenbank
 - Laufzeit

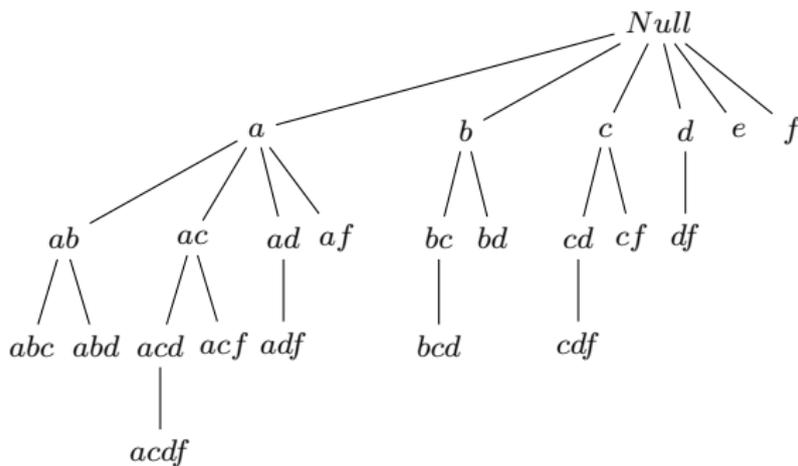
R.C. Agarwal, C.C. Aggarwal, V.V. Prasad 2001

„A Tree Projection Algorithm for Generation of Frequent Itemsets“

in: J. Parallel and Distribute Computing 61(3), 350 – 371

LTrees

- Lexikografische Ordnung der items – hier:
 $a < b < c < d < e < f$
- Mögliche Erweiterungen eines Knotens P sind die lexikalisch größeren Geschwister – hier:
 $R(a) = \{b, c, d, e, f\}$,
 $R(ab) = \{c, d, f\}$,
 $R(b) = \{c, d\}$, $R(bc) = \{d\}$
- Erweiterung eines Knotens P um ein weiteres häufiges item hier: $E(a) = \{b, c, d, f\}$,
 $E(b) = \{c, d\}$,
 $E(c) = \{d, f\}$, $E(d) = \{f\}$
- $E(P) \subseteq R(P) \subseteq E(Q)$
 wobei P eine Erweiterung von Q ist.

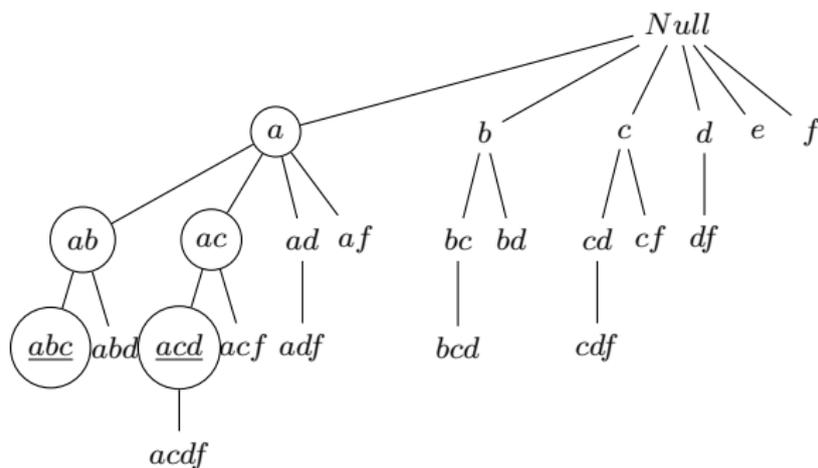


Angenommenes Beispiel A



LTrees in frequent set mining

- Knoten ist aktiv, wenn er als Erweiterung generiert wird – hier:
 $\{a, ab, ac, abc, acd\}$;
inaktiv, wenn der Baum, dessen Wurzel er ist, nicht erweitert werden kann.
- Ein Grenzknoten ist ein aktiver Knoten, dessen Erweiterung noch nicht generiert ist – hier:
 $\{abc, acd\}$.
- Aktive items $F(P)$ von P sind
 - P ist Grenzknoten, dann $F(P) = R(P)$
 - Sonst aktive Knoten in $E(P)$ und deren aktive items.

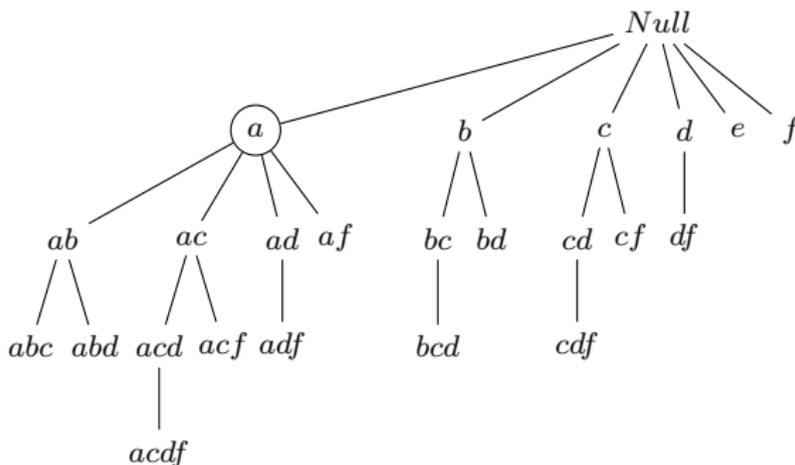




Knoten

● An einem Knoten P sind gespeichert:

- Der item set P
- $AE(P)$: Aktuell aktive Erweiterungen von P
- $F(P)$: aktive items von P
- Matrix $E(P) \times E(P)$



a

$$AE(a) = \{b, c\}$$

$$F(a) = \{b, c, d, f\}$$

a	b	c	d	f
b	-			
c	# abc	-		
d	# abd	# acd	-	
f	# abf	# acf	# adf	-

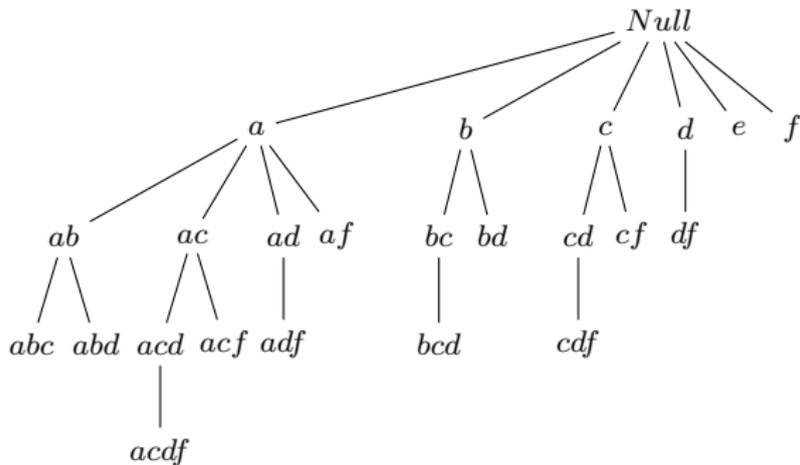


Tree Projection

- Projektion von Datenbanktupel T auf itemset P :

- falls $T \cap P = \{\}$, ist $T(P) = null$,
- sonst:
 $T(P) = T \cap F(P)$

- Beispiel: Transaktion $\{a, b, c, d, e, f, g, h, k\}$
 $T(a) = \{b, c, d, f\}$





Algorithmus – Breitensuche

- Aufbau des LTrees
 - *AddTree()*
 - *PruneTree()*
- Tree projection
 - *AddCounts()*
- Vorteile Breitensuche:
 - Beschränkung auf eine Ebene in einem Schritt → passt in Hauptspeicher
- Nachteile Breitensuche:
 - Tree projection auf jeder Ebene → k DB scans

BreadthFirst($minSup : s, DB : T$)

$L_1 :=$ all frequent 1-itemsets;

$E(null) :=$ set of items in L_1 ;

make top-level of LTree;

$k := 1$;

while level – k not null do

- create matrices at level $k - 1$ nodes
- *for each T in T do*
AddCounts(T);
- *AddTree(k); // creates L_{k+1}*
- *PruneTree(k); // deletes inactive nodes up to level $k + 1$*
- $k := k + 1$;



AddTree(), *PruneTree()*

- *AddTree(k)*
 - L_{k+1} = alle $k + 1$ itemsets mit ausreichendem support;
 - Knoten auf Ebene $k + 1$ hinzufügen;
- *PruneTree(k)*
 - entferne alle inaktiven Knoten auf Ebene $k + 1$;
 - für jeden Knoten P auf Ebene $k + 1$ do $F(P) := R(P)$;
 - for $r = k, - - 1, \text{until } 0$ do
 - entferne inaktive Knoten auf Ebene r ;
 - update $F(P)$ aller Knoten auf Ebene r und ihrer Kinder;

Matrix zählen

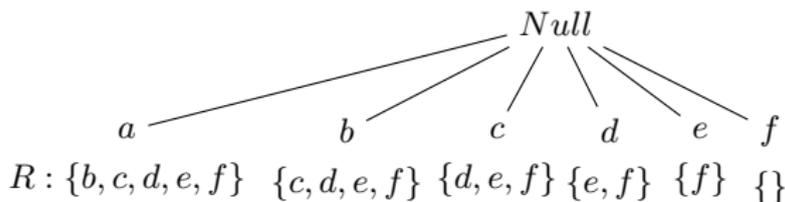
Beispiel Matrix

$E(Null) \times E(Null)$

für Kandidaten der Ebene 2:

Verarbeitung von 4

Transaktionen



Transaktionen

	a	b	c	d	e
b	ab 1				
c	ac 2	bc 1			
d	ad 2	bd 2	cd 3		
e	ae 1	be 2	ce 2	de 3	
f	af 2	bf 1	cf 3	df 3	ef 2

Matrix zählen

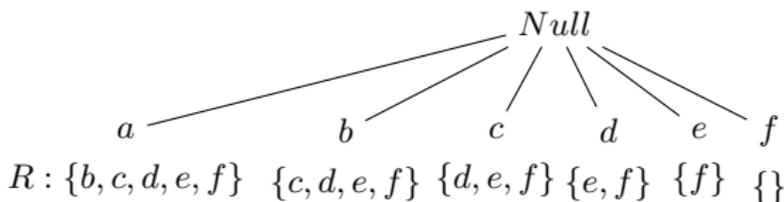
Beispiel Matrix

$E(Null) \times E(Null)$

für Kandidaten der Ebene 2:

Verarbeitung von 4

Transaktionen



	a	b	c	d	e
b	ab 1				
c	ac 2	bc 1			
d	ad 2	bd 2	cd 3		
e	ae 1	be 2	ce 2	de 3	
f	af 2	bf 1	cf 3	df 3	ef 2

Transaktionen

acdf

Matrix zählen

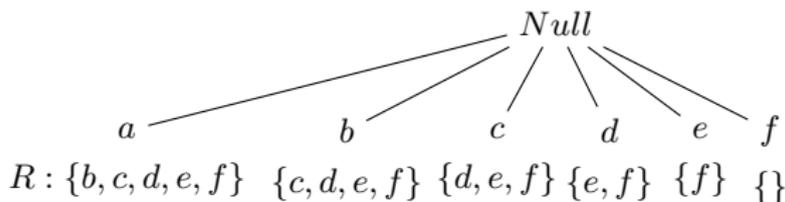
Beispiel Matrix

$E(Null) \times E(Null)$

für Kandidaten der Ebene 2:

Verarbeitung von 4

Transaktionen



	a	b	c	d	e
b	ab 1				
c	ac 2	bc 1			
d	ad 2	bd 2	cd 3		
e	ae 1	be 2	ce 2	de 3	
f	af 2	bf 1	cf 3	df 3	ef 2

Transaktionen

acdf

abcdef

Matrix zählen

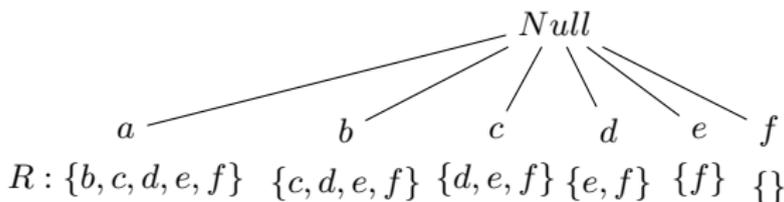
Beispiel Matrix

$E(Null) \times E(Null)$

für Kandidaten der Ebene 2:

Verarbeitung von 4

Transaktionen



	a	b	c	d	e
b	ab 1				
c	ac 2	bc 1			
d	ad 2	bd 2	cd 3		
e	ae 1	be 2	ce 2	de 3	
f	af 2	bf 1	cf 3	df 3	ef 2

Transaktionen

- acdf
- abcdef
- bde

Matrix zählen

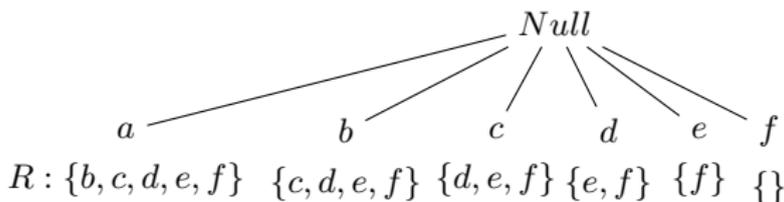
Beispiel Matrix

$E(Null) \times E(Null)$

für Kandidaten der Ebene 2:

Verarbeitung von 4

Transaktionen



	a	b	c	d	e
b	ab 1				
c	ac 2	bc 1			
d	ad 2	bd 2	cd 3		
e	ae 1	be 2	ce 2	de 3	
f	af 2	bf 1	cf 3	df 3	ef 2

Transaktionen

- acdf
- abcdef
- bde
- cdef



Projektion

- Transaktionen müssen auf die Grenzknoten projiziert werden, wo dann in der Matrix gezählt wird.
- Beim Erzeugen von Knoten auf Ebene $k + 1$, wird für alle Knoten P der Ebene $k - 1$ jeweils eine Matrix $E(P) \times E(P)$ angelegt.
- Was dann noch an Speicher frei ist, ist für die Transaktionen da.
- Strategien:
 - Je 1 Transaktion auf alle Knoten der Ebene $k - 1$ projizieren: weniger Speicher, mehr Rechenzeit;
 - Viele Transaktionen auf einen Knoten projizieren: bessere Rechenzeit, mehr Speicher.
 - Ausweg: Transaktion von oben nach unten über die Ebenen projizieren, blockweise.

Cache-Blöcke

- Zählen der Häufigkeit von $k + 1$ -itemsets, die Nachfolger von $k - 1$ -Knoten sind gemäß Ausschnitten aus der Matrix.
- Beispiel: Transaktion $\{a, b, c, d, e, f\}$ strip, c'

| | |
 $p1$ $p2$ $p3$

Auf jede Transaktion, die mindestens ein item mit denen im ,strip' teilt, werden 3 Zeiger gesetzt.

	a	b	strip c	d	e
b	ab				
c	ac	bc			
d	ad	bd	cd+1		
e	ae	be	ce+1	de+1	
f	af	bf	cf+1	df+1	ef+1

for *outerP* $p1$ bis $p2$ do

for *innerP* = *outerP* + 1 to $p3$ do

*Matrix*Eintrag(*outerP*, *innerP*) + 1



Tiefensuche

- Vorteile Tiefensuche

- von der Wurzel zum aktuellen Knoten wird die Baumprojektion einfach durchgereicht

- Nachteile Tiefensuche

- passt am Anfang nicht in Hauptspeicher, denn vom Wurzelknoten wird die gesamte Datenbank hinunterprojiziert

- Kombination:

- anfangs Breitensuche
- sobald alle Baumprojektionen an den Grenzknoten in den Hauptspeicher passen, werden sie in separate Dateien je Grenzknoten gespeichert und jeweils per Tiefensuche bearbeitet.



Experimente

- Sehr effizient: 213 972 Transaktionen mit durchschnittlich 31 items
 - level 0 (2-itemset Kandidaten) 23,49 CPU Sekunden bei 16 276 365 Matrixeinträgen
 - level 1 (3-itemset Kandidaten) 25,44 CPU Sekunden bei 3 521 972 Matrixeinträgen
 - level 2 (4-itemset Kandidaten) 9,76 CPU Sekunden bei 219 269 Matrixeinträgen
- Puffer von Transaktionen wird depth-first auf Knoten im Cache projiziert – günstige Zugriffe auf immer die selben Adressen im Cache.



Was wissen wir jetzt?

- LTrees sind kompakter als hash trees.
- Tree Projection verwendet LTrees für häufige Mengen – lexikographische Ordnung.
- Es gibt keine explizite Kandidatengenerierung, aber der Aufbau des LTrees wird ähnlich wie bei Apriori realisiert. Allerdings wird erst hinterher der Baum gestutzt. Daher langsamer als FP growth.
- Sorgfalt bei der Speicherausnutzung:
 - Breiten- vs. Tiefensuche beim LTree-Aufbau,
 - Tiefensuche beim Projizieren der Tupel
 - Ausschnitte aus den Matrizen nach ‚strips‘