Vorlesung
Wissensentdeckung in Datenbanken

Häufige Mengen: Apriori und FP-Growth

Kristian Kersting, (Katharina Morik), Claus Weihs

LS 8 Informatik
Computergestützte Statistik
Technische Universität Dortmund

17.4.2014

Was geschah in der letzten Vorlesung?

- Wir haben SQL wiederholt und sind insbesondere auf Aggregation eingegangen
- Sie haben den Data Cube kennengelernt: Aggregation über p Dimensionen
- Sie haben gesehen, wie mit mit seiner Hilfe exploratives Data Mining betreiben kann: OLAP (Online Analytical Processing)
- Wir haben gesehen, dass die Berechnung eine Data Cubes ein interessantes Optimierungsproblem ist. Fragestellungen des Data Mining, der Datenbanktheorie und der Architektur gehen oft Hand in Hand, gerade bei Big Data.

Heute: Häufige Mengen automatisch finden

- OLAP ist Benutzer bezogen: muss man wissen, wonach man "sucht"
- Wenn viele Relationen vorliegen, kann es schwierig sein, den Überblick zu behalten

Wie finden wir automatisch Muster in einer Datenbank, von denen wir vielleicht nicht einmal geträumt haben ?

- Häufige Mengen
- Assoziationsregeln

Gliederung

Lernen von Assoziationsregeln

Gegeben:

- R eine Menge von Objekten, die binäre Werte haben
- t eine Transaktion, $t \subseteq R$
- r eine Menge von Transaktionen
- $s_{min} \in [0, 1]$ die minimale Unterstützung (support),
- $conf_{min} \in [0, 1]$ die minimale Konfidenz (confidence)

Finde alle Regeln c der Form $X \rightarrow Y$, wobei $X \subseteq R$, $Y \subseteq R$, $X \cap Y = \{\}$

$$\text{Unterstützung} \quad s(r, c) = \frac{|\{t \in r \mid X \cup Y \subseteq t\}|}{|r|} \geq s_{min} \tag{1}$$

$$\text{Konfidenz} \quad conf(r, c) = \frac{|\{t \in r \mid X \cup Y \subseteq t\}|}{|\{t \in r \mid X \subseteq r\}|} \geq conf_{min} \tag{2}$$

Binäre Datenbanken

Sei R eine Menge von Objekten, die binäre Werte haben, und r eine Menge von Transaktionen, dann ist $t \in r$ eine Transaktion und die Objekte mit dem Wert 1 sind eine Teilmenge aller Objekte.

$$R = \{A, B, C\}$$

$$t = \{B, C\} \subseteq R$$

A	B	C	ID
0	1	1	1
1	1	0	2
0	1	1	3
1	0	0	4

Warenkorbanalyse (Market Basket Analysis)

Aftershave	Bier	Chips	EinkaufsID
0	1	1	1
1	1	0	2
0	1	1	3
1	0	0	4

$\{\text{Aftershave}\} \rightarrow \{\text{Bier}\}$ $\quad s = \frac{1}{4}, conf = \frac{1}{2}$

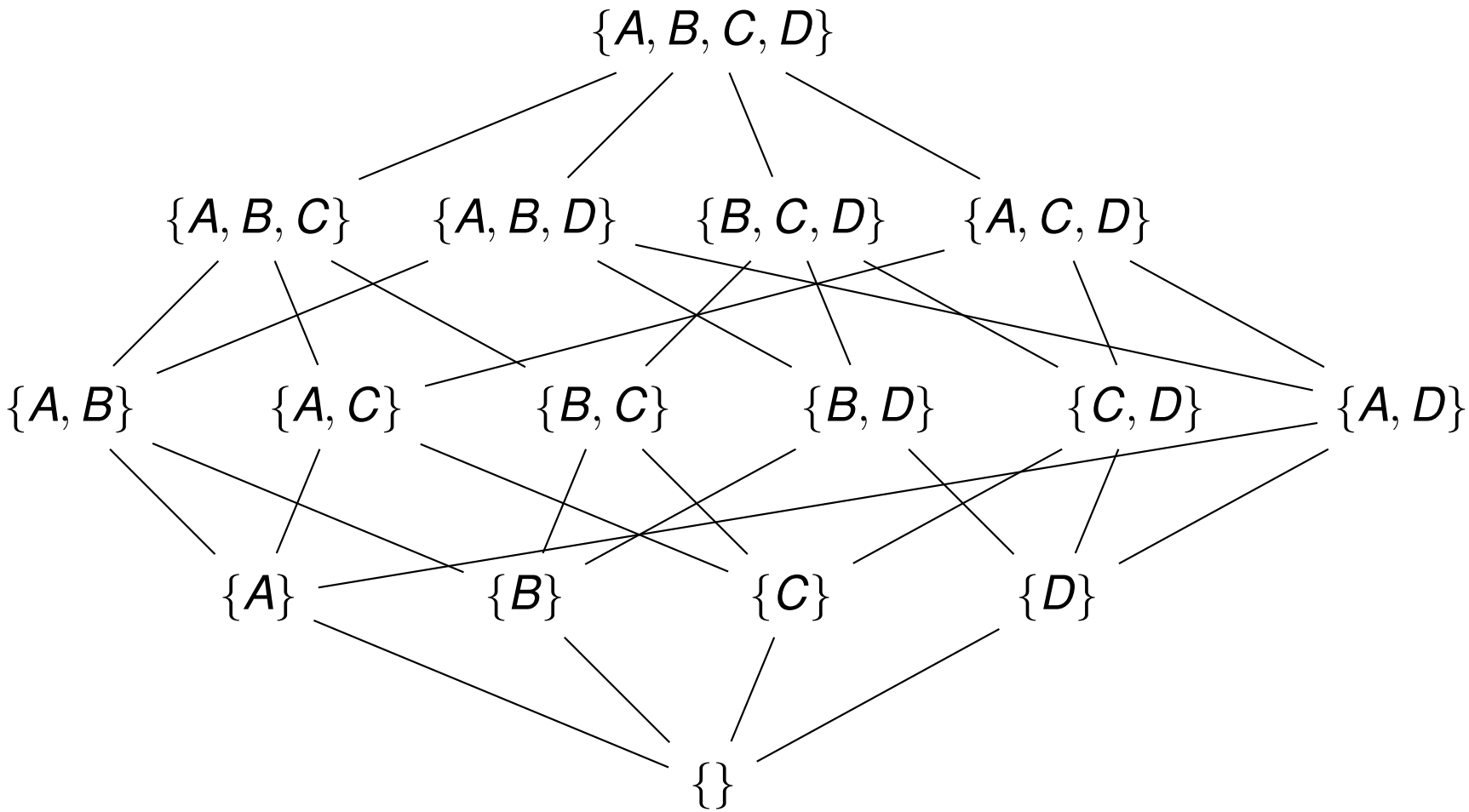
$\{\text{Aftershave}\} \rightarrow \{\text{Chips}\}$ $\quad s = 0$

$\{\text{Bier}\} \rightarrow \{\text{Chips}\}$ $\quad s = \frac{1}{2}, conf = \frac{2}{3}$ (zusammen anbieten?)

$\{\text{Chips}\} \rightarrow \{\text{Aftershave}\}$ $\quad s = 0$

$\{\text{Aftershave}\} \rightarrow \{\text{Bier, Chips}\}$ $\quad s = 0$

Verband (Lattice)



Ordnungsrelation

- Hier ist die Ordnungsrelation die Teilmengenbeziehung.
- Eine Menge S_1 ist größer als eine Menge S_2, wenn $S_1 \supseteq S_2$.
- Eine kleinere Menge ist allgemeiner.

Assoziationsregeln

Achtung: Assoziationsregeln sind keine logischen Regeln!

- In der Konklusion können mehrere Attribute stehen
- Attribute sind immer nur binär.
- Mehrere Assoziationsregeln zusammen ergeben kein Programm.

Achtung: Binärvektoren (Transaktionen)

- Attribute sind eindeutig geordnet.

Aufgabe:

- Aus häufigen Mengen Assoziationsregeln herstellen

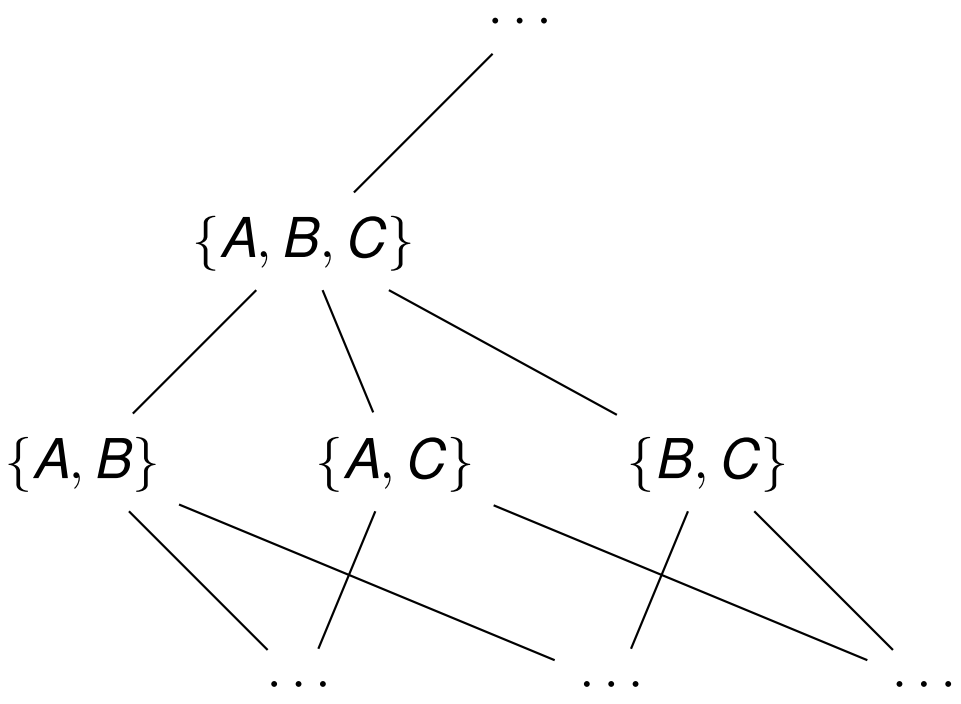
Apriori Algorithmus
(Agrawal, Mannila, Srikant, Toivonen, Verkamo 1996)

Häufige Mengen $L_k = X \cup Y$ mit k Objekten (frequent itemsets)
(X und Y beziehen sich hier auf die Assoziationregeln!)

- Wenn eine Menge häufig ist, so auch all ihre Teilmengen. (Anti-Monotonie)
- Wenn eine Menge selten ist, so auch all ihre Obermengen. (Monotonie)
- Wenn X in L_{k+1} ist, dann sind alle $S_i \subseteq X$ in L_k (Anti-Monotonie)
- Alle Mengen L_k, die $k-1$ Objekte gemeinsam haben, werden vereinigt zu C_{k+1}. Das ist die Menge der Kandidaten für L_{k+1}.

Dies ist der Kern des Algorithmus, die Kandidatengenerierung.

Beispiel



- Wenn $\{A, B, C\}$ häufig ist, dann sind auch $\{A, B\}$, $\{A, C\}$, $\{B, C\}$ häufig.
- Das bedeutet, daß$\{A, B\}$, $\{A, C\}$, $\{B, C\}$ ($k = 2$) häufig sein müssen, damit $\{A, B, C\}$ ($k + 1 = 3$) häufig sein *kann*.
- Also ergeben die häufigen Mengen aus L_k die Kandidaten C_{k+1}

Wir können häufige Mengen level-weise berechnen.

Beispiele

Gesucht werden Kandidaten mit $k + 1 = 5$

$L_4 = \{\{ABCD\}, \{ABCE\}, \{ABDE\}, \{ACDE\}, \{BCDE\}\}$

- $k - 1$ Stellen gemeinsam vereinigen zu: $l = \{ABCDE\}$
- Sind alle k langen Teilmengen von l in L_4?
  $\{ABCD\}\{ABCE\}\{ABDE\}\{ACDE\}\{BCDE\}$ - ja!
- Dann wird l Kandidat C_5.

$L4 = \{\{ABCD\}, \{ABCE\}\}$
$l = \{ABCDE\}$

- Sind alle Teilmengen von l in L_4?
  $\{ABCD\}\{ABCE\}\{ABDE\}\{ACDE\}\{BCDE\}$ - nein!
- Dann wird l nicht zum Kandidaten.

Kandidatengenerierung

- Erzeuge-Kandidaten(L_k)
  - $C_{k+1} := \{\}$
  - Für alle l_1, l_2 in L_k, so dass

$$l_1 = \{i_1, \cdots, i_{k-1}, i_k\} \text{ und}$$

$$l_2 = \{i_1, \cdots, i_{k-1}, i'_k\} \text{ mit } i'_k < i_k$$

    - $l := \{i_1, \cdots, i_{k-1}, i_k, i'_k\}$
    - falls alle k-elementigen Teilmengen von l in L_k sind, dann

$$C_{k+1} := C_{k+1} \cup \{l\}$$

    - return C_{k+1}
- Prune(C_{k+1}, r) vergleicht Häufigkeit von Kandidaten mit s_{min}.

Häufige Mengen

- Häufige-Mengen(R, r, s_{min})
  - $C_1 := \cup_{i \in R} i$,
  - Setze $k = 1$
  - $L_1 :=$ Prune(C_1)
  - while $L_k \neq \{\}$
    - $C_{k+1} :=$ Erzeuge-Kandidaten(L_k)
    - $L_{k+1} :=$ Prune(C_{k+1}, r)
    - $k := k + 1$
    - return $\cup_{j=2}^{k} L_j$

APRIORI

- Apriori(R, r, s_{min}, $conf_{min}$)
  - L :=Häufige-Mengen(R, r, s_{min})
  - c :=Regeln (L, $conf_{min}$)
  - return c

Aber wie werden die Regeln generiert?

Regelgenerierung

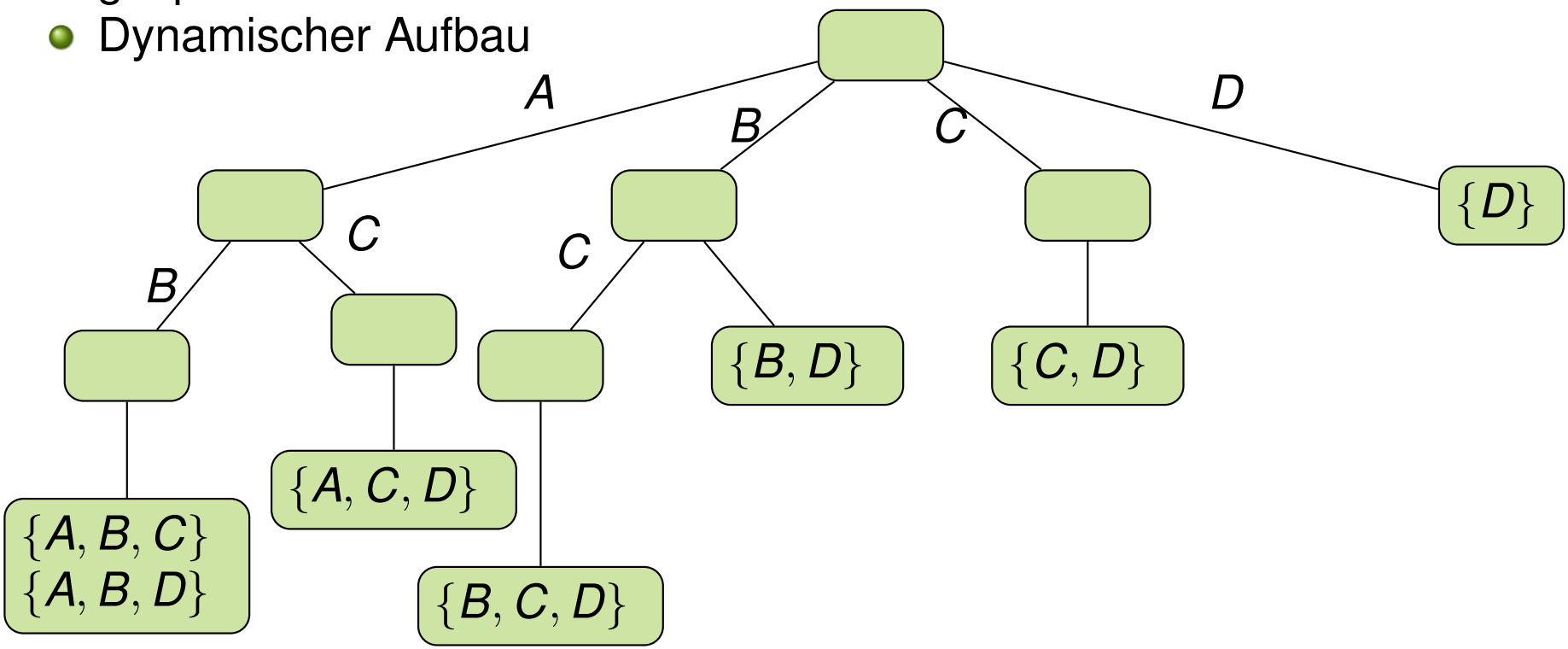
Aus den häufigen Mengen werden Regeln geformt. Wenn die Konklusion länger wird, kann die Konfidenz sinken. Die Ordnung der Attribute wird ausgenutzt:

$$
\begin{array}{lll}
l_1 = \{i_1, \cdots, i_{k-1}, i_k\} & c_1 = \{i_1, \cdots, i_{k-1}\} \rightarrow \{i_k\} & conf_1 \\
l_1 = \{i_1, \cdots, i_{k-1}, i_k\} & c_2 = \{i_1, \cdots\} \rightarrow \{i_{k-1}, i_k\} & conf_2 \\
\cdots & \cdots & \cdots \\
l_1 = \{i_1, \cdots, i_{k-1}, i_k\} & c_k = \{i_1\} \rightarrow \{\cdots, i_{k-1}, i_k\} & conf_k
\end{array}
$$

$$conf_1 \geq conf_2 \geq \cdots \geq conf_k$$

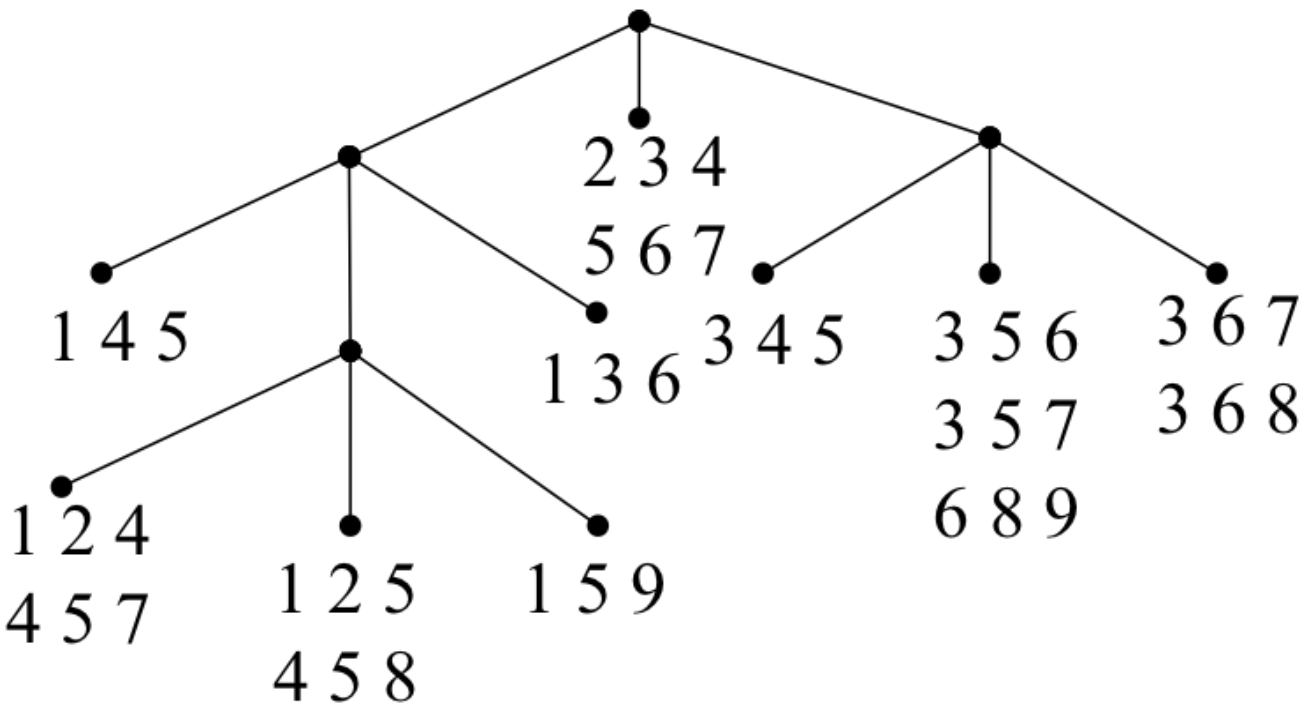
Implementierung: Präfixbaum

- Die Zahl der Vergleiche zur Berechnung der Unterstützung kann sehr groß werden !
- Daher brauchen wir effizientere Strukturen, um die Zahl der Vergleiche zu reduzieren: Hash-Tree oder Präfixbaum (Trie), der sich aus der Ordnung der Elemente in den Mengen ergibt.
- An jedem Knoten werden die Schlüssel und/oder Häufigkeit gespeichert.
- Dynamischer Aufbau



Implementierung: Hash-Tree

- Nehmen wir mal an, dass wir
  $\{1,4,5\}, \{1,2,4\}, \{4,5,7\}, \{1,2,5\}, \{4,5,8\}, \{1,5,9\}, \{1,3,6\}$,
  $\{2,3,4\}, \{5,6,7\}, \{3,4,5\}, \{3,5,6\}, \{3,5,7\}, \{6,8,9\}$,
  $\{3,6,7\}, \{3,6,8\}$ hätten.
- Die Hash-Funktion bildet jetzt jeden Wert auf $0, 1, 2$ ab, also z.B. $f(x) = x \ mod \ 3$. 1 steht für "links", 2 für "mitte" und 0 für "rechts.



Was wissen Sie jetzt?

- Assoziationsregeln sind keine logischen Regeln.
- Anti-Monotonie der Häufigkeit: Wenn eine Menge häufig ist, so auch all ihre Teilmengen.
- Man erzeugt häufige Mengen, indem man häufige Teilmengen zu einer Menge hinzufügt und diese Mengen dann auf Häufigkeit testet. Bottom-up Suche im Verband der Mengen.(Generierung der Kandidaten)
- Monotonie der Seltenheit: Wenn eine Teilmenge selten ist, so auch jede Menge, die sie enthält.
- Man beschneidet die Suche, indem Mengen mit einer seltenen Teilmenge nicht weiter betrachtet werden. (Pruning)

Probleme von Apriori

- Im schlimmsten Fall ist Apriori exponentiell in R, weil womöglich alle Teilmengen gebildet würden. In der Praxis sind die Transaktionen aber spärlich besetzt. Die Beschneidung durch s_{min} und $conf_{min}$ reicht bei der Warenkorbanalyse meist aus.
- Apriori liefert unglaublich viele Regeln.
- Die Regeln sind höchst redundant.
- Die Regeln können irreführend sein, weil die Kriterien die a priori Wahrscheinlichkeit nicht berücksichtigen. Wenn sowieso alle Cornflakes essen, dann essen auch hinreichend viele Fußballer Cornflakes.

Prinzipien für Regelbewertungen

1. $RI(A \to B) = 0$, wenn $|A \to B| = \frac{(|A||B|)}{|r|}$
   A und B sind unabhängig.
2. $RI(A \to B)$ steigt monoton mit $|A \to B|$.
3. $RI(A \to B)$ fällt monoton mit $|A|$ oder $|B|$.

Also:

- $RI > 0$, wenn $|A \to B| > \frac{(|A||B|)}{|r|}$, d.h. wenn A positiv mit B korreliert ist.
- $RI < 0$, wenn $|A \to B| < \frac{(|A||B|)}{|r|}$, d.h. wenn A negativ mit B korreliert ist.

Wir wissen, dass immer $|A \to B| \leq |A| \leq |B|$ gilt, also

- RI_{min}, wenn $|A \to B| = |A|$ oder $|A| = |B|$
- RI_{max}, wenn $|A \to B| = |A| = |B|$

Piatetsky-Shapiro 1991

Konfidenz

- Die Konfidenz erfüllt die Prinzipien nicht! (Nur das 2.) Auch unabhängige Mengen A und B werden als hoch-konfident bewertet.
- Die USA-Census-Daten liefern die Regel

$$\text{aktiv-militär} \rightarrow \text{kein-Dienst-in-Vietnam}$$

mit 90% Konfidenz. Tatsächlich ist $s(\text{kein-Dienst-in-Vietnam}) = 95\%$ Es wird also wahrscheinlicher, wenn aktiv-militär gegeben ist!

- Gegeben eine Umfrage unter 2000 Schülern, von denen 60% Basketball spielen, 75% Cornflakes essen. Die Regel

$$\text{Basketball} \rightarrow \text{Cornflakes}$$

hat Konfidenz 66% Tatsächlich senkt aber Basketball die Cornflakes Häufigkeit!

Signifikanztest

- Ein einfaches Maß, das die Prinzipien erfüllt, ist:

$$|A \to B| - \frac{|A||B|}{|r|}$$

- Die Signifikanz der Korrelation zwischen A und B ist:

$$\frac{|A \to B| - \frac{|A||B|}{|r|}}{\sqrt{|A||B|\left(1 - \frac{A}{r}\right)\left(1 - \frac{|B|}{|r|}\right)}}$$

Sicherheitsmaß

Shortliffe, Buchanan 1990 führten ein Sicherheitsmaß CF ein (für Regeln in Wissensbasen)

- Wenn $conf(A \rightarrow B) > s(B)$
  $CF(A \rightarrow B) = conf(A \rightarrow B) - \frac{s(B)}{1-s(B)}$
- Wenn $conf(A \rightarrow B) < s(B)$
  $CF(A \rightarrow B) = conf(A \rightarrow B)$
- Sonst
  $CF(A \rightarrow B) = 0$

Das Sicherheitsmaß befolgt die Prinzipien für Regelbewertung. Wendet man Signifikanztest oder Sicherheitsmaß an, erhält man weniger (irrelevante, irreführende) Assoziationsregeln.

Was wissen Sie jetzt?

- Sie haben drei Prinzipien für die Regelbewertung kennengelernt:
  - Unabhängige Mengen sollen mit 0 bewertet werden.
  - Der Wert soll höher werden, wenn die Regel mehr Belege hat.
  - Der Wert soll niedriger werden, wenn die Mengen weniger Belege haben.
- Sie haben drei Maße kennen gelernt, die den Prinzipien genügen:
  - Einfaches Maß
  - statistisches Maß und
  - Sicherheitsmaß

FP-Tree: Finden von häufige Mengen ohne Kandidatengenerierung (Jiawei Han and Micheline Kamber, 2000)

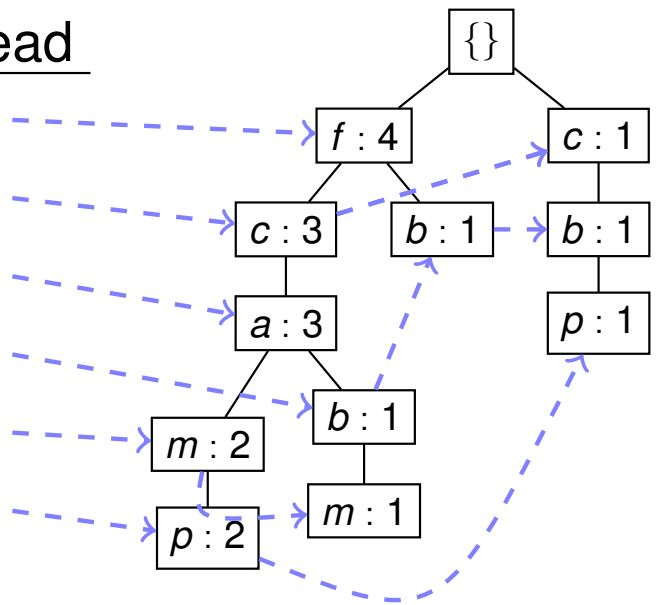
- Idee: Komprimiere die Daten in eine kompakte Datenstruktur: Frequent-Pattern tree (FP-Tree)
  - sehr verdichtet, aber vollständig fürs Finden von häufigen Mengen
  - Vermeidet teure Scans der Daten
- Basierend auf FP-Trees werden dann häufige Mengen und somit Assoziationsregeln gefunden
  - Teile-und-Herrsche Ansatz: Die Data Mining Aufgaben werden in kleinere Aufgaben aufgeteilt
  - Kandidatengenerierung wird vermieden, in dem nur die "Sub-Datenbanken" getestet werden

Konstruktion eines FP-trees aus einer DB von Transaktionen

TID	Items bought	(ordered) frequent items
100	$\{f, a, c, d, g, i, m, p\}$	$\{f, c, a, m, p\}$
200	$\{a, b, c, f, l, m, o\}$	$\{f, c, a, b, m\}$
300	$\{b, f, h, j, o\}$	$\{f, b\}$
400	$\{b, c, k, s, p\}$	$\{c, b, p\}$
500	$\{a, f, c, e, l, p, m, n\}$	$\{f, c, a, m, p\}$

$support_{min} = 0.5$

Item	*freq*	head
f	4	
c	4	
a	3	
b	3	
m	3	
p	3	



1. Scan die DB einmal, um die häufige 1-itemsets zu finden (single item pattern)
2. Ordne die häufigen Mengen absteiget nach Häufigkeit
3. Scan die DB nochmals, um den FP-tree aufzubauen

Was ist ein *FP*-Tree ?

- Ein *FP* Tree ist nach Häufigkeiten (von oben nach unten) geordnet.
- Ein *FP* Tree fasst Transaktionen als Wörter auf und stellt gemeinsame Präfixe verschiedener Wörter dar.
- Für jede Transaktion lege einen Pfad im *FP* Tree an:
  - Pfade mit gemeinsamem Präfix - Häufigkeit $+1$, Suffix darunter hängen.
  - Kein gemeinsamer Präfix vorhanden - neuen Zweig anlegen.
- Header Tabelle verweist auf das Vorkommen der Items im Baum. Auch die Tabelle ist nach Häufigkeit geordnet.
- Und weil wir einen Präfix-Baum haben, verweisen wir "von links nach rechts" auf das nächste Vorkommen des Wortes im Baum.

Vorteile der *FP*-Tree Struktur

- Vollständigkeit:
  - Alle langen Muster in Transaktionen bleiben erhalten.
  - Jegliche Information relevant fürs Finden der häufigen Mengen bleibt erhalten.
- Kompaktheit:
  - Reduktion der irrelevanten Informationen — seltene Items sind fallen weg
  - Wegen der Sortierung nach absteigende Häufigkeit werden häufige Items gemeinsam “benutze”.
  - Niemals größer als die ursprüngliche DB (wenn man von Knoten-Links und Counts absieht)
  - Für einige DBs ist die Kompressionsrate über 100.

OK, jetzt haben wir FP-Trees. Und wie finden wir jetzt die Häufigen Mengen?

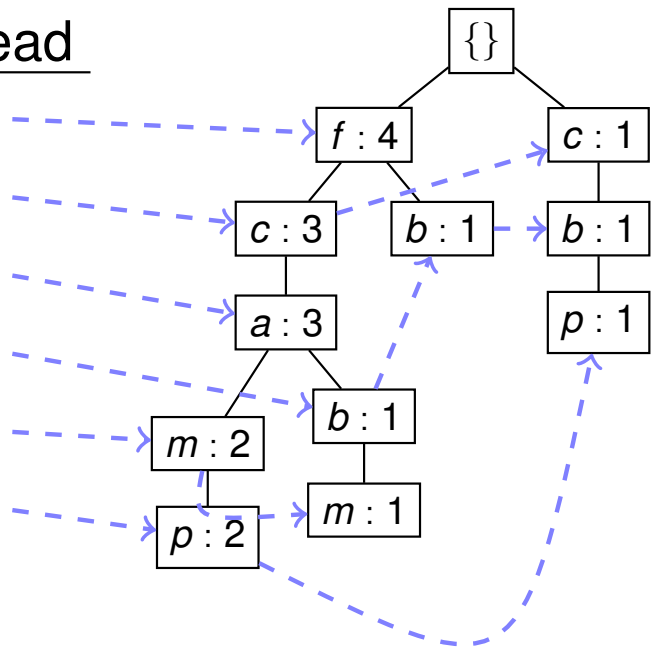
Mining Häufige Muster mittels *FP*-Trees: *FP*-Growth

- Generelle Idee (Teile-und-Herrsche)
  - Baue rekursiv “häufige Muster”-Pfade mittels des FP-Trees.
- Methode
  - Bestimme für jedes Item seine bedingte Musterbank (*conditional pattern-base*) und seinen bedingten *FP*-Tree.
  - Wiederhole rekursive dies auf den neu hinzugekommenen bedingten *FP*-Trees solange,
  - bis der resultierende *FP*-Tree leer oder nur noch einen Pfad enthält. Diese Pfad generiert all Kombinationen seiner Teilpfade, die alle häufige Mengen/Muster sind. Diese zählt man einfach auf.

Schritt 1: Vom *FP*-tree zur Bedingten Musterbank

- Fange mit der Header-Tabelle als Verweise in den *FP*-Tree an.
- Gehe den *FP*-Tree für alle häufigen Items durch.
- Füge alle entsprechenden Präfix-Pfade in die Bedingte Musterbank ein. Die Häufigkeiten werden von den "Einstiegspunkten" übernommen.

Item	*freq*	head
f	4	
c	4	
a	3	
b	3	
m	3	
p	3	

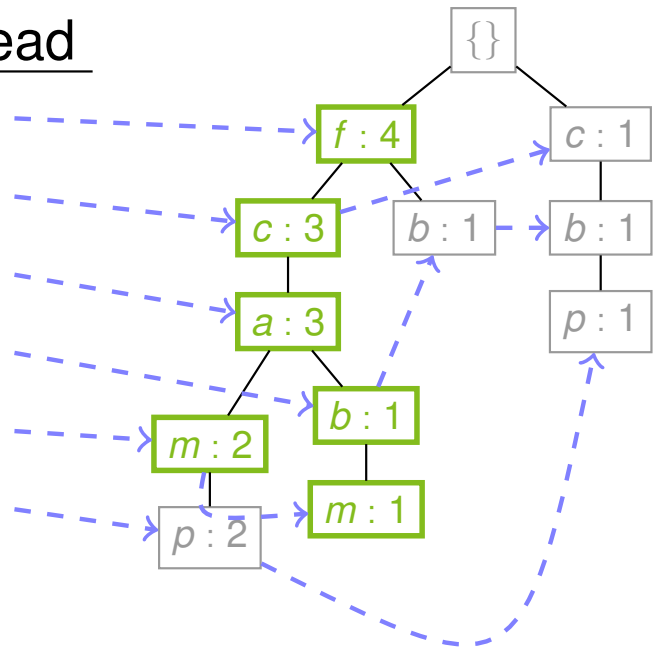


Item	conditional pattern base
c	*f* : 3
a	*fc* : 3
b	*fca* : 1, *f* : 1, *c* : 1
m	*fca* : 2, *fcab* : 1
p	*fcam* : 2, *cb* : 1

Schritt 2: Konstruktion der Bedingten (Conditional) *FP*-trees

- Für jede Musterbank
  - Akkumuliere die Counts (Zählungen) für jedes Item in der Bank.
  - Konstruiere den *FP*-Tree für die häufigen Mengen in der Musterbank.

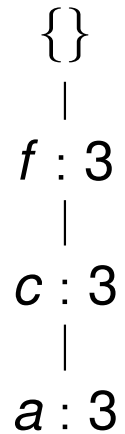
Item	*freq*	head
f	4	
c	4	
a	3	
b	3	
m	3	
p	3	



m-conditional pattern base:

fca : 2, *fcab* : 1

m-conditional *FP*-tree:



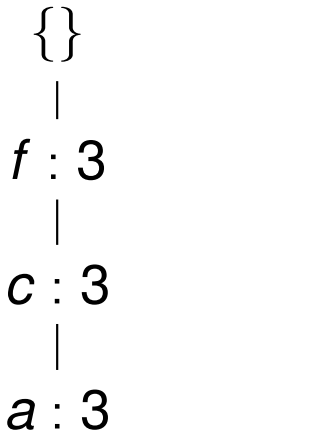
Von der Cond. Pattern Base zum Cond. *FP* Tree

- Präfixpfade eines Suffixes bilden die bedingte Basis.
- Diejenigen Präfixpfade, die häufiger als $support_{min}$ sind, bilden den bedingten *FP* Tree.
- Falls mehrere dieser Präfixpfade zu einem Suffix gleich sind (vom Anfang bis zu einer bestimmten Stelle), wird daraus ein Pfad bis zu dieser Stelle und die ursprünglichen Häufigkeiten werden addiert.
- Ansonsten gibt es mehrere Pfade im bedingten Baum.

Vom *m*-conditional *FP*-tree zu den häufigen Mengen

m-conditional
pattern base:
fca : 2, *fcab* : 1

m-conditional *FP*-tree:



$\Longrightarrow$

Alle häufigen Mengen für *m*

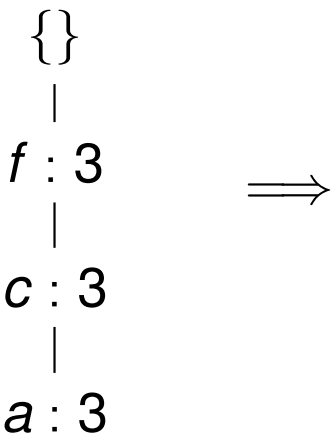
- *m*
- *fm*, *cm*, *am*
- *fcm*, *fam*, *cam*
- *fcam*

Häufige Mengen mittels Bedingten Musterbanken

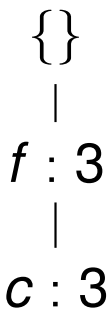
Item	Conditional pattern-base	conditional *FP*-tree
p	$\{(fcam:2),(cb:1)\}$	$\{(c:3)\}\|p$
m	$\{(fca:2),(fcab:1)\}$	$\{(f:3,c:3,a:3)\}\|m$
b	$\{(fca:1),(f:1),(c:1)\}$	Empty
a	$\{(fc:3)\}$	$\{((f:3,c:3)\}\|a$
c	$\{(f:3)\}$	$\{(f:3)\}\|c$
f	Empty	Empty

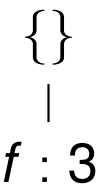
Schritt 3: Rekursive Berechnung der bedingten *FP*-Trees

	Cond. pattern base of "*am*": (*fc* : 3)	Cond. pattern base of "*cm*": (*f* : 3)	Cond. pattern base of "*cam*": (*f* : 3)
m-conditional *FP*-tree:	*am*-conditional *FP*-tree:	*cm*-conditional *FP*-tree:	*cam*-conditional *FP*-tree:



$\Longrightarrow$





Rekursionsabbruch: Einzelner Pfad im *FP*-tree

- Annahme: nur ein einzige Pfad P in einem (bedingten) FP-tree T.
- Alle häufigen Mengen von T können durch das Aufzählen aller möglichen Teilpfade von P generiert werden.

m-conditional *FP*-tree:

$\{\}$
|
$f : 3$
|
$c : 3$
|
$a : 3$

$\Longrightarrow$

Alle häufigen Mengen für m

- m
- fm, cm, am
- fcm, fam, cam
- $fcam$

Alle häufigen Mengen

- Die gesuchte Menge der häufigen Mengen ist die Gesamtheit alles häufiger Muster aus allen bedingten *FP* Bäumen.

Und warum funktioniert das? Das Prinzip von (Frequent Pattern) *FP*- Growth

- Pattern-Growth Eigenschaft
  - Sei α eine Häufige Menge in *DB*, *B* die α's bedingte Musterbank und β eine Itemmenge in *B*. Dann ist $\alpha \cup \beta$ eine häufige Menge in *DB* gdw. β häufig in *B* ist.
- Beispiel: "*abcdef*" ist eine häufige Menge gdw.
  - "*abcde*" ist eine häufige Menge und
  - "*f*" kommt häufig in den Transaktionen mit "*abcde*" vor.

Algorithmus *FP_growth*

Input:

- D eine Transaktionsdatenbank
- $support_{min}$ ein Schwellwert der Häufigkeit

1. Scan von D, Erstellen der Menge F häufiger Items und ihrer Häufigkeiten, Ordnen von F in absteigender Häufigkeit.
2. Wurzel des *FP* Trees ist Null. Für jede Transaktion Trans in D: nach Häufigkeit gemäßF geordnete Items in Trans werden zur Liste $[p|P]$, wobei p das erste item und P die restlichen sind. *insert_tree*$([p|P], T)$
3. *FP_growth*$(FP\_tree, null)$

insert_tree($[p|P], T$)

- Wenn T ein Kind N hat mit $N.item\_name = p.item\_name$ dann erhöhe Häufigkeit von $N + 1$.
- Sonst bilde neuen Knoten N mit Häufigkeit $= 1$ direkt unter T und füge Knotenverweise zu den Knoten mit dem selben *item.name* ein.
- Solange P nicht $\{\}$ ist, *insert_tree*(P, N).

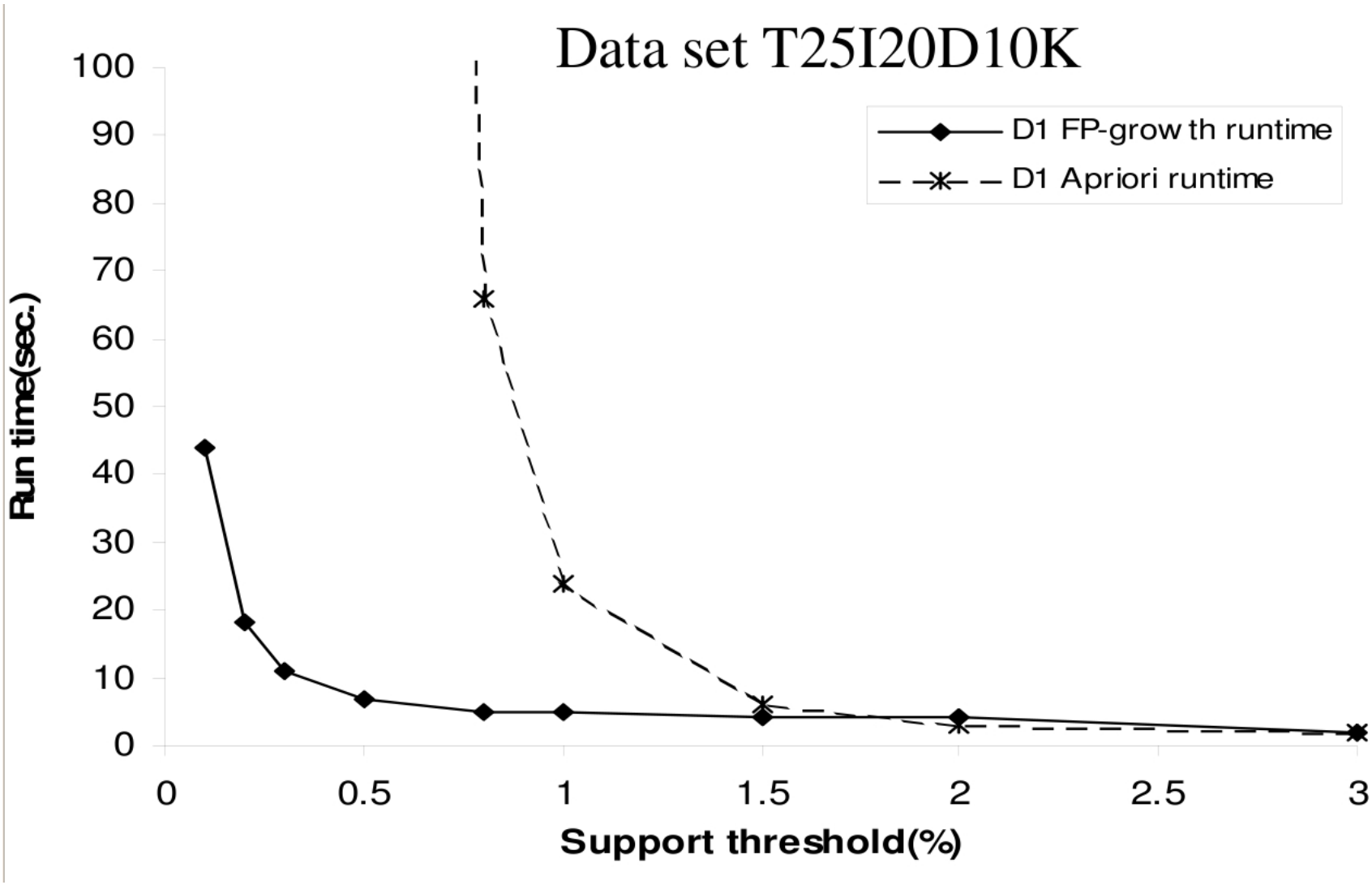
fp_growth(*Tree*, α)

- Wenn Tree ein einziger Pfad P ist,
  - dann generiere für jede Kombination β von Knoten in P Muster $\beta \cup \alpha$ mit $support = support_{min}$ eines Items in β.
- Sonst für jedes a_i in header von Tree
  - generiere Muster $\beta = a_i \cup \alpha$ mit $s = a_i.s$
  - konstruiere β cond. base und daraus β cond. *FP* tree $Tree_\beta$
  - Wenn $Tree_\beta$ nicht $\{\}$, dann *fp_growth*($Tree_\beta, \beta$)

Warum is Frequent Pattern Growth schnell?

- Empirische Untersuchungen zeigen, dass
  - *FP*-growth um eine Größenordnung schneller sein kann als Apriori.
- Begründung
  - Keine Kandidatengenerierung, keine Kandidatentests !
  - Ausnutzung einer kompakten Datenstruktur.
  - Reduktion der vollständigen DB-Durchläufe/Scans.
  - Einfache Basisoperationen: zählen und *FP*-Tree Aufbauen.

FP-growth vs. Apriori: Skalierung mit dem Support-Threshold



Was wissen wir jetzt?

- *FP*-growth als Alternative zu Apriori
  - Schneller, weil keine Kandidaten generiert werden
  - Kompaktes Speichern
  - Basisoperation ist einfach Zählen.
- Der *FP*-Baum gibt Präfixbäume für ein Suffix an.
- Die Ordnungsrelation ist die Häufigkeit der Items.
  - Der Baum wird vom häufigsten zum seltensten gebaut.
  - Die bedingte Basis wird vom seltensten Suffix zum häufigsten erstellt.

Insgesamt, interessante Zusammenhänge in Transaktionen können automatisch und effizient gefunden werden (z.B. mittels Apriori bzw. FP-Growth).