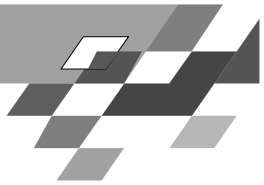
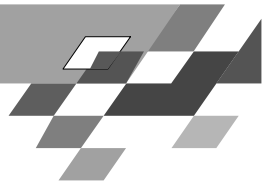


Data Cube

1. Einführung
2. Aggregation in SQL, GROUP BY
3. Probleme mit GROUP BY
4. Der Cube-Operator
5. Implementierung des Data Cube
6. Zusammenfassung und Ausblick

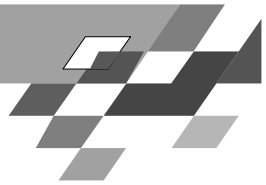
Dank an Hanna Köpcke!



On-line Analytical Processing (OLAP)

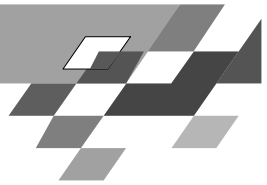
Ziel: Auffinden interessanter Muster in großen Datenmengen

- Formulierung einer Anfrage
- Extraktion der Daten
- Visualisierung der Ergebnisse
- Analyse der Ergebnisse und Formulierung einer neuen Anfrage



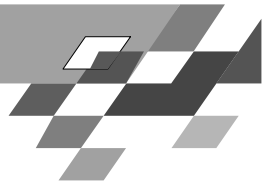
OLAP-Werkzeuge

- Datenmenge wird als n-dimensionaler Raum aufgefasst
- Identifizierung von „interessanten" Unterräumen
- In relationalen Datenbanken werden n-dimensionale Daten als Relationen mit n-Attributen modelliert
- Dimensionsreduktion durch Aggregation der Daten entlang der weggelassenen Dimensionen



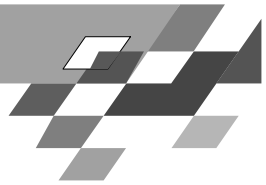
Beispiel: Autoverkäufe

Modell	Jahr	Farbe	Anzahl
Opel	1990	rot	5
Opel	1990	weiß	87
Opel	1990	blau	62
Opel	1991	rot	54
Opel	1991	weiß	95
Opel	1991	blau	49
Opel	1992	rot	31
Opel	1992	weiß	54
Opel	1992	blau	71
Ford	1990	rot	64
Ford	1990	weiß	62
Ford	1990	blau	63
Ford	1991	rot	52
Ford	1991	weiß	9
Ford	1991	blau	55
Ford	1992	rot	27
Ford	1992	weiß	62
Ford	1992	blau	39



Aggregation in SQL

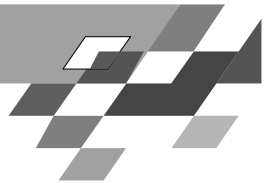
- Aggregatfunktionen:
  COUNT(), SUM(), MIN(), MAX(), AVG()
  Beispiel: SELECT AVG(Anzahl)
  FROM Autoverkäufe
- Aggregation über verschiedene Werte
  Beispiel: SELECT COUNT(DISTINCT Modell)
  FROM Autoverkäufe
- Aggregatfunktionen liefern einen einzelnen Wert
- Aggregation über mehrere Attribute mit GROUP BY



GROUP BY

```
SELECT Modell, Jahr, SUM(Anzahl)
    FROM Autoverkäufe
    GROUP BY Modell, Jahr
```

- Die Tabelle wird gemäß den Kombinationen der ausgewählten Attributmenge in Gruppen unterteilt
- Jede Gruppe wird über eine Funktion aggregiert
- Das Resultat ist eine Tabelle mit aggregierten Werten, indiziert durch die ausgewählte Attributmenge

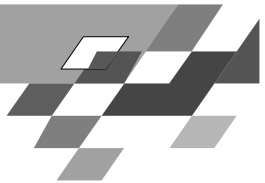


Beispiel: GROUP BY

Modell	Jahr	Farbe	Anzahl
Opel	1990	rot	5
Opel	1990	weiß	87
Opel	1990	blau	62
Opel	1991	rot	54
Opel	1991	weiß	95
Opel	1991	blau	49
Opel	1992	rot	31
Opel	1992	weiß	54
Opel	1992	blau	71
Ford	1990	rot	64
Ford	1990	weiß	62
Ford	1990	blau	63
Ford	1991	rot	52
Ford	1991	weiß	9
Ford	1991	blau	55
Ford	1992	rot	27
Ford	1992	weiß	62
Ford	1992	blau	39

```
SELECT Modell, Jahr, SUM(Anzahl)
FROM Autoverkäufe
GROUP BY Modell, Jahr
```

Modell	Jahr	Anzahl
Opel	1990	154
Opel	1991	198
Opel	1992	156
Ford	1990	189
Ford	1991	116
Ford	1992	128



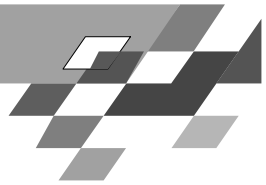
Roll Up

Gleiche Anfrage in unterschiedlichen Detailierungsgraden

- Verminderung des Detailierungsgrades = Roll Up
- Erhöhung des Detailierungsgrades = Drill Down

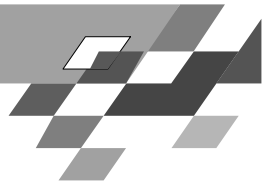
Beispiel: Autoverkäufe

- Roll Up über drei Ebenen
- Daten werden nach Modell, dann nach Jahr, dann nach Farbe aggregiert
- die Verkaufszahlen werden zuerst für jedes Modell aus jedem Jahr in jeder Farbe aufgelistet, dann werden alle Verkaufszahlen des gleichen Modells und Jahres aufsummiert und daraus die Verkaufszahlen der Modelle berechnet



GROUP BY: Roll Up

Modell	Jahr	Farbe	Anzahl nach Modell, Jahr, Farbe	Anzahl nach Modell, Jahr	Anzahl nach Modell
Opel	1990	rot	5		
		weiß	87		
		blau	62		
				154	
	1991	rot	54		
		weiß	95		
		blau	49		
				198	
	1992	rot	31		
		weiß	54		
		blau	71		
				156	
					508

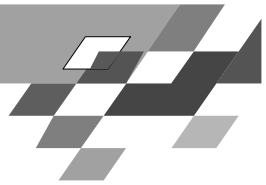


Probleme mit GROUP BY: Roll Up

- Tabelle ist nicht relational, da man wegen der leeren Felder (Null-Werte) keinen Schlüssel festlegen kann.
- Die Zahl der Spalten wächst mit der Zahl der aggregierten Attribute
- Um das exponentielle Anwachsen der Spaltenanzahl zu vermeiden, wird der ALL-Wert eingeführt.
- Der ALL-Wert repräsentiert die Menge, über die die Aggregation berechnet wird.

Beispiel:

Ein ALL in der Spalte Farbe bedeutet, dass in der Anzahl dieser Zeile die Verkaufszahlen der roten, weißen und blauen Autos zusammengefasst sind.

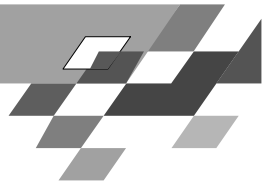


GROUP BY: Roll Up mit ALL

Modell	Jahr	Farbe	Anzahl
Opel	1990	rot	5
Opel	1990	weiß	87
Opel	1990	blau	62
Opel	1990	ALL	154
Opel	1991	rot	54
Opel	1991	weiß	95
Opel	1991	blau	49
Opel	1991	ALL	198
Opel	1992	rot	31
Opel	1992	weiß	54
Opel	1992	blau	71
Opel	1992	ALL	156
Opel	ALL	ALL	506

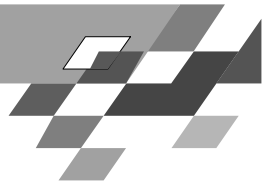
Erzeugung der Tabelle mit SQL:

```
SELECT Modell, ALL, ALL, SUM(Anzahl)
  FROM Autoverkäufe
  WHERE Modell = 'Opel'
  GROUP BY Modell
UNION
  SELECT Modell, Jahr, ALL, SUM(Anzahl)
  FROM Autoverkäufe
  WHERE Modell = 'Opel'
  GROUP BY Modell, Jahr
UNION
  SELECT Modell, Jahr, Farbe, SUM(Anzahl)
  FROM Autoverkäufe
  WHERE Modell = 'Opel'
  GROUP BY Modell, Jahr, Farbe
```



Probleme mit GROUP BY: Roll Up

- Beispiel war ein einfaches dreidimensionales Roll Up
- Eine Aggregation über n Dimensionen erfordert n Unions
- Roll Up ist asymmetrisch:
  Verkäufe sind nach Jahr, aber nicht nach Farbe aggregiert



Kreuztabellen

Symmetrische Darstellung mehrdimensionaler Daten und Aggregationen

Opel	1990	1991	1992	Total (ALL)
rot	5	54	31	90
weiß	87	95	54	236
blau	62	49	71	182
Total (ALL)	154	198	156	508

Diese Kreuztabelle ist eine zweidimensionale Aggregation

Nimmt man noch andere Automodelle hinzu, kommt für jedes Modell eine weitere Ebene hinzu

Man erhält eine dreidimensionale Aggregation

Der CUBE-Operator

n-dimensionale Generalisierung der bisher genannten Konzepte

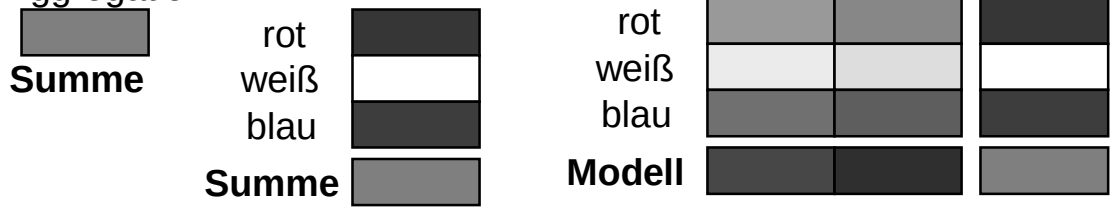
Der 0D Data Cube ist ein Punkt

Der 1D Data Cube ist eine Linie mit einem Punkt

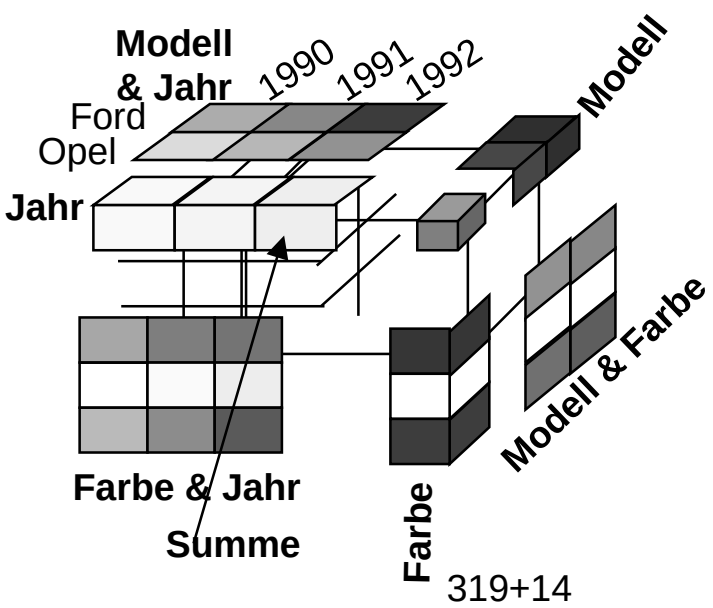
Der 2D Data Cube ist eine Kreuztabelle

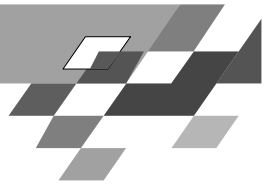
Der 3D Data Cube ist ein Würfel mit drei sich überschneidenden Kreuztabellen

(Gray, Chaudhuri, Bosworth, Layman 1997)



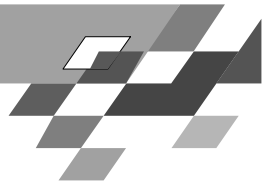
Data Cube mit allen Aggregationen





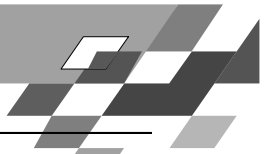
Der CUBE-Operator

- Beispiel: **SELECT** Modell, Jahr, Farbe, **SUM**(Anzahl)
  FROM Autoverkäufe
  GROUP BY CUBE Modell, Jahr, Farbe
- Der Cube-Operator erzeugt eine Tabelle, die sämtliche Aggregationen enthält
- Es werden GROUP BYs für alle möglichen Kombinationen der Attribute berechnet
- Die Erzeugung der Tabelle erfordert die Generierung der Potenzmenge der zu aggregierenden Spalten.
- Bei n Attributen werden 2^n GROUP BYs berechnet
- Sei $C_1, C_2, ..., C_n$ die Kardinalität der n Attribute, dann ist die Kardinalität der resultierenden Data Cube-Relation $\prod(C_i + 1)$

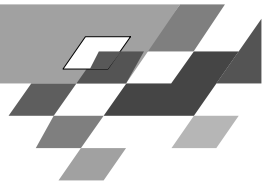


Data Cube des Beispiels

Modell	Jahr	Farbe	Anzahl
Opel	1990	rot	5
Opel	1990	weiß	87
Opel	1990	blau	62
Opel	1991	rot	54
Opel	1991	weiß	95
Opel	1991	blau	49
Opel	1992	rot	31
Opel	1992	weiß	54
Opel	1992	blau	71
Ford	1990	rot	64
Ford	1990	weiß	62
Ford	1990	blau	63
Ford	1991	rot	52
Ford	1991	weiß	9
Ford	1991	blau	55
Ford	1992	rot	27
Ford	1992	weiß	62
Ford	1992	blau	39

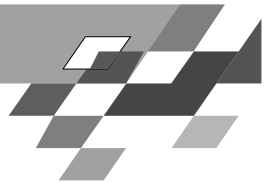


Modell	Jahr	Farbe	Anzahl
Opel	1990	rot	5
Opel	1990	weiß	87
Opel	1990	blau	62
Opel	1990	ALL	154
Opel	1991	rot	54
Opel	1991	weiß	95
Opel	1991	blau	49
Opel	1991	ALL	198
Opel	1992	rot	31
Opel	1992	weiß	54
Opel	1992	blau	71
Opel	1992	ALL	156
Opel	ALL	rot	90
Opel	ALL	weiß	236
Opel	ALL	blau	182
Opel	ALL	ALL	508
Ford	1990	rot	64
Ford	1990	weiß	72
Ford	1990	blau	63
Ford	1990	ALL	189
Ford	1991	rot	52
Ford	1991	weiß	9
Ford	1991	blau	55
Ford	1991	ALL	116
Ford	1992	rot	27
Ford	1992	weiß	62
Ford	1992	blau	39
Ford	1992	ALL	128
Ford	ALL	rot	143
Ford	ALL	weiß	133
Ford	ALL	blau	157
Ford	ALL	ALL	433
ALL	1990	rot	69
ALL	1990	weiß	149
ALL	1990	blau	125
ALL	1990	ALL	343
ALL	1991	rot	106
ALL	1991	weiß	104
ALL	1991	blau	104
ALL	1991	ALL	314
ALL	1992	rot	58
ALL	1992	weiß	116
ALL	1992	blau	110
ALL	1992	ALL	284
ALL	ALL	rot	233
ALL	ALL	weiß	369
ALL	ALL	blau	339
ALL	ALL	ALL	941



Implementationsalternativen

- Physische Materialisierung des gesamten Data Cube:
  - beste Antwortzeit
  - hoher Speicherplatzbedarf
- Keine Materialisierung:
  - jede Zelle wird nur bei Bedarf aus den Rohdaten berechnet
  - kein zusätzlicher Speicherplatz
  - schlechte Antwortzeit
- Materialisierung von Teilen des Data Cube:
  - Werte vieler Zellen sind aus Inhalt anderer Zellen berechenbar
  - diese Zellen nennt man „abhängige" Zellen
  - Zellen, die einen All-Wert enthalten, sind abhängig
  - Problem: Welche Zellen des Data Cube materialisieren?
  - Zellen des Data Cube entsprechen SQL Anfragen (Sichten)

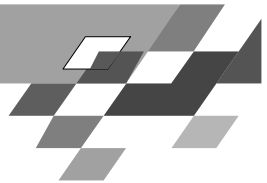


Abhängigkeit von Sichten

Die Abhängigkeitsrelation $\leq$ zwischen zwei Anfragen Q_1 und Q_2

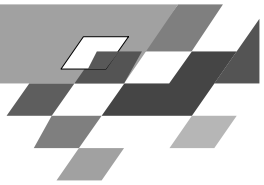
$Q_1 \leq Q_2$ gdw. Q_1 kann beantwortet werden, indem die Ergebnisse von Q_2 verwendet werden. Q_1 ist abhängig von Q_2

- Anfragen bilden einen Verband unter folgenden Voraussetzungen:
  1. $\leq$ ist eine Halbordnung und
  2. es gibt ein maximales Element (eine oberste Sicht)
- Der Verband wird durch eine Menge von Anfragen (Sichten) L und der Abhängigkeitsrelation $\leq$ definiert und mit $\langle L, \leq \rangle$ bezeichnet
- Ein Verband wird dargestellt durch einen Graphen, in dem die Anfragen die Knoten sind und $\leq$ die Kanten.



Auswahl von Sichten zur Materialisierung

- Optimierungsproblem, das unter folgenden Bedingungen gelöst werden soll:
  - Die durchschnittliche Zeit für die Auswertung der Anfragen soll minimiert werden.
  - Man beschränkt sich auf eine feste Anzahl von Sichten, die materialisiert werden sollen, unabhängig von deren Platzbedarf
- Das Optimierungsproblem ist NP-vollständig.
- Heuristiken für Appoximationslösungen:
  Greedy-Algorithmus
- Der Greedy-Algorithmus verhält sich nie zu schlecht:
  Man kann zeigen, dass die Güte mindestens 63% beträgt.
  (Harinayaran, Rajaraman, Ullman 1996)



Der Greedy Algorithmus

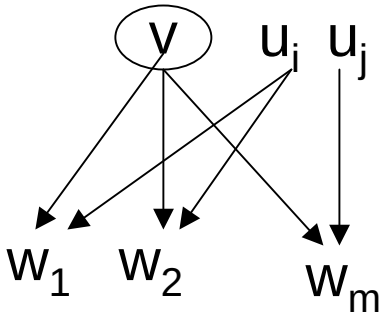
- Gegeben ein Verband mit Speicherkosten C(v) für jede Sicht v
- Annahme: Speicherkosten = Anzahl der Reihen in der Sicht
- Beschränkung auf k materialisierte Sichten
- Nach Auswahl einer Menge S von Sichten wird der Nutzen der Sicht v relativ zu S mit B(v, S) bezeichnet und wie folgt definiert:

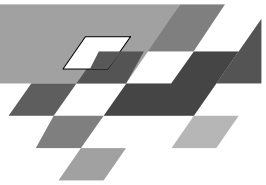
  1. Für jede Sicht $w \leq v$ wird B_w berechnet:

     (a) Sei u die Sicht mit den geringsten Kosten in S, so dass $w \leq u$

     (b) $B_w = \begin{cases} C(u) - C(v), & \text{falls } C(v) < C(u) \\ 0 & \text{ansonsten} \end{cases}$

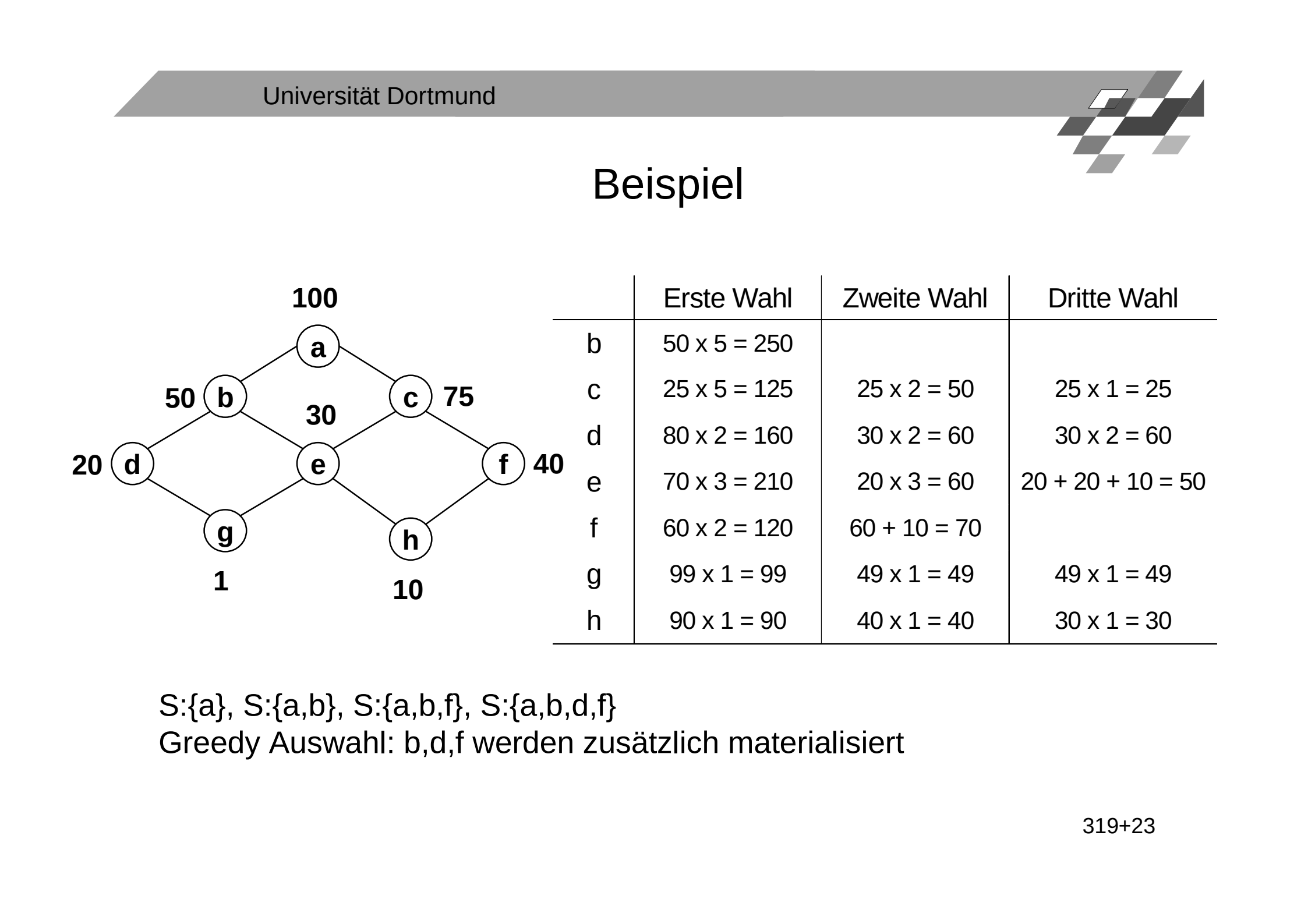
  2. $B(v, S) = \sum_{w \leq v} B_w$





Der Greedy Algorithmus

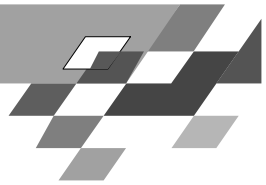
```
1  S = {oberste Sicht}
2  for i = 1 to k do begin
3       Wähle die Sicht v∉ S, so dass B(v, S) maximal ist;
4       S = S ∪ {v}
5       end;
6  return S;
```



Beispiel

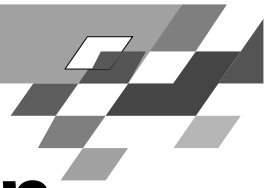
	Erste Wahl	Zweite Wahl	Dritte Wahl
b	50 x 5 = 250		
c	25 x 5 = 125	25 x 2 = 50	25 x 1 = 25
d	80 x 2 = 160	30 x 2 = 60	30 x 2 = 60
e	70 x 3 = 210	20 x 3 = 60	20 + 20 + 10 = 50
f	60 x 2 = 120	60 + 10 = 70	
g	99 x 1 = 99	49 x 1 = 49	49 x 1 = 49
h	90 x 1 = 90	40 x 1 = 40	30 x 1 = 30

S:{a}, S:{a,b}, S:{a,b,f}, S:{a,b,d,f}
Greedy Auswahl: b,d,f werden zusätzlich materialisiert



Was wissen Sie jetzt?

- Möglichkeiten und Grenzen der Aggregation in SQL
- Einführung von Data Cubes zur Unterstützung von Aggregationen über n Dimensionen
- Greedy-Algorithmus zur Auswahl einer festen Anzahl von Sichten, die materialisiert werden



Lernen von Assoziationsregeln

Gegeben:

R eine Menge von Objekten, die binäre Werte haben

t eine Transaktion, $t \subseteq R$

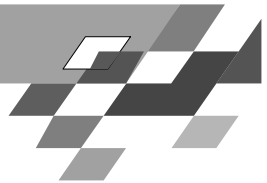
r eine Menge von Transaktionen

Smin $\in$ [0,1] die minimale Unterstützung,

Confmin $\in$ [0,1] die minimale Konfidenz

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

$$s(r,c) = \frac{\left|\left\{ t \in r \middle| X \cup Y \in t \right\}\right|}{|r|} \geq s_{\min} \qquad conf(r,c) = \frac{\left|\left\{ t \in r \middle| X \cup Y \in t \right\}\right|}{\left|\left\{ t \in r \middle| X \in r \right\}\right|} \geq conf_{\min}$$



Binäre Datenbanken

R eine Menge von Objekten, die binäre Werte haben

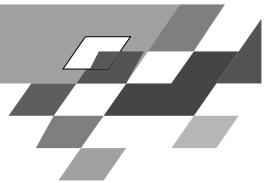
A, B, C

r eine Menge von Transaktionen

t eine Transaktion, $t \subseteq R$

B,C

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

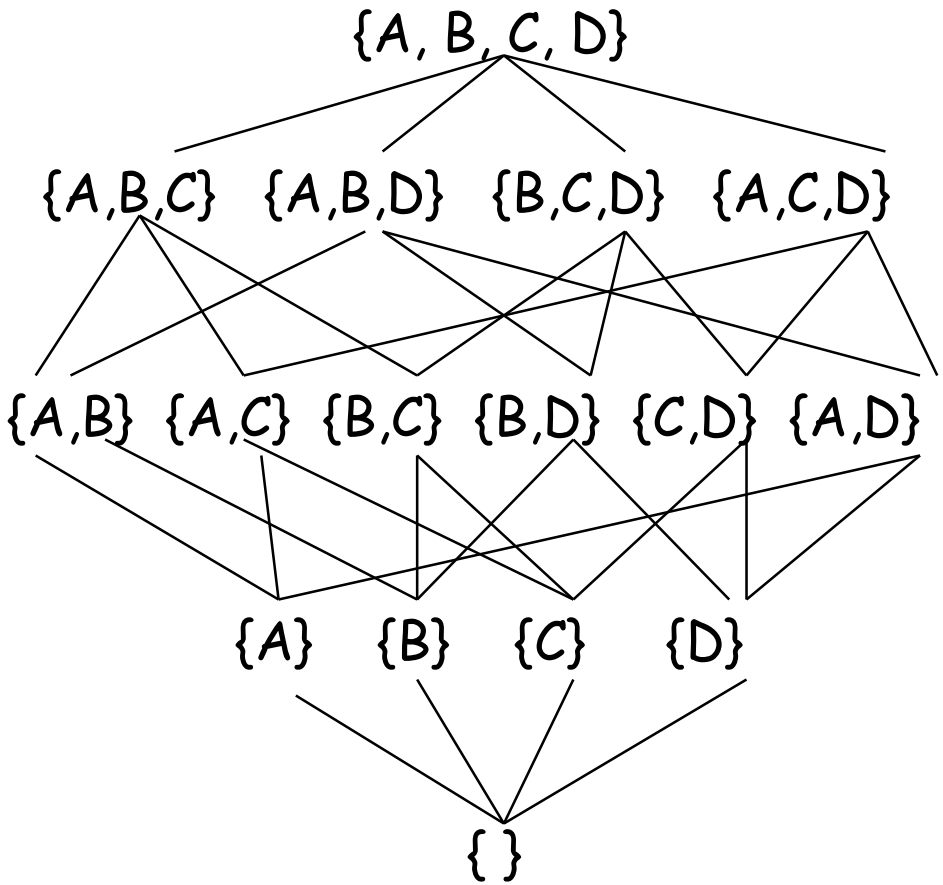


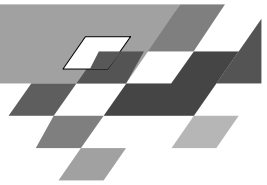
Warenkorbanalyse

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

{Aftershave}→{Bier} $s = \frac{1}{4}$, conf = $\frac{1}{2}$
{Aftershave}→ {Chips} $s = 0$
{Bier} → {Chips} $s = \frac{1}{2}$, conf= 2/3 -- zusammen anbieten?
{Chips}→{Aftershave} $s = 0$
{Aftershave}→{Bier,Chips} $s = 0$

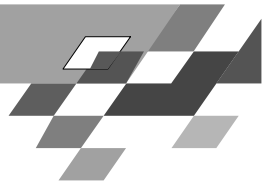
Wieder ein Verband...





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

LH: Assoziationsregeln sind keine logischen Regeln!

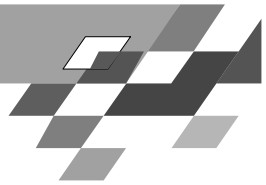
- In der Konklusion können mehrere Attribute stehen
- Attribute sind immer nur binär.
- Mehrere Assoziationsregeln zusammen ergeben kein Programm.

LE: Binärvektoren (Transaktionen)

- Attribute sind eindeutig geordnet.

Aufgabe:

- Aus häufigen Mengen Assoziationsregeln herstellen



Apriori Algorithmus

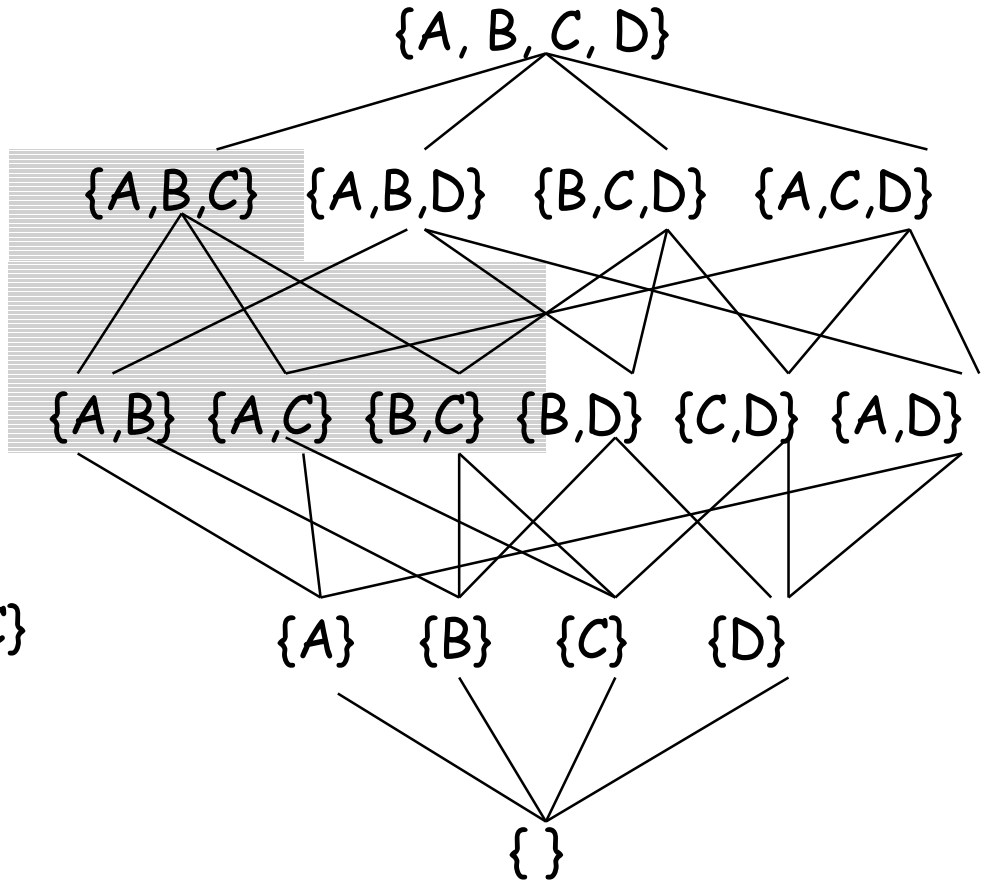
(Agrawal, Mannila, Srikant, Toivonen, Verkamo 1996)

LH des Zwischenschritts: Häufige Mengen $L_k = X \cup Y$
mit k Objekten (large itemsets, frequent sets)

- 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} dann alle $S_i \subseteq X$ in L_k (Anti-Monotonie)
- Alle Mengen L_k, die k-1 Objekte gemeinsam haben, werden vereinigt zu L_{k+1}.

Dies ist der Kern des Algorithmus', die Kandidatengenerierung.

Beispiel



Wenn häufig

k+1=3

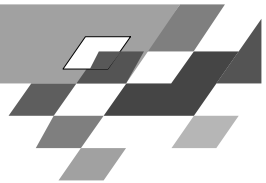
dann häufig

k=2

Generiere aus
{A,B},{A,C},{B,C}

{A,B,C}

Häufige Mengen L_k ergeben Kandidaten C_{k+1}



Beispiel

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.

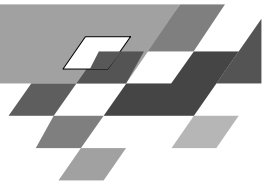
L_4= { {ABCD}, {ABCE} }

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)

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

Forall l_1, l_2 in L_k , sodass $l_1 = \{i_1, ..., i_{k-1}, i_k\}$

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

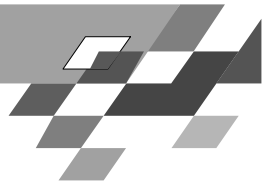
$l := \{i_1, ..., i_{k-1}, i_k, i'_k\}$

if alle k-elementigen Teilmengen von l in L_k sind

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

Return C_{k+1}

Prune(C_{k+1}, r) vergleicht Häufigkeit von Kandidaten mit smin.

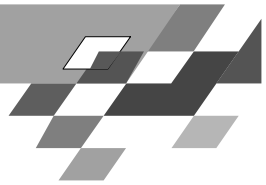


Häufige Mengen

Häufige-Mengen(R, r, smin)

$C_1 := \bigcup_{i \in R} \{i\}$, k=1,
$L_1 := \text{Prune}(C_1)$
while $L_k \neq \{\}$
    $C_{k+1} := \text{Erzeuge-Kandidaten}(L_k)$
    $L_{k+1} := \text{Prune}(C_{k+1}, r)$
    $k := k+1$

Return $\bigcup_{j=2}^{k} L_j$



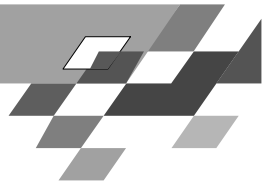
APRIORI

Apriori(R, r, smin, confmin)

L:= Häufige-Mengen(R, r, smin)

c:= Regeln (L, confmin)

Return c.

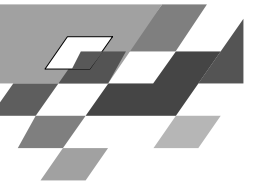


Regelgenerierung

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

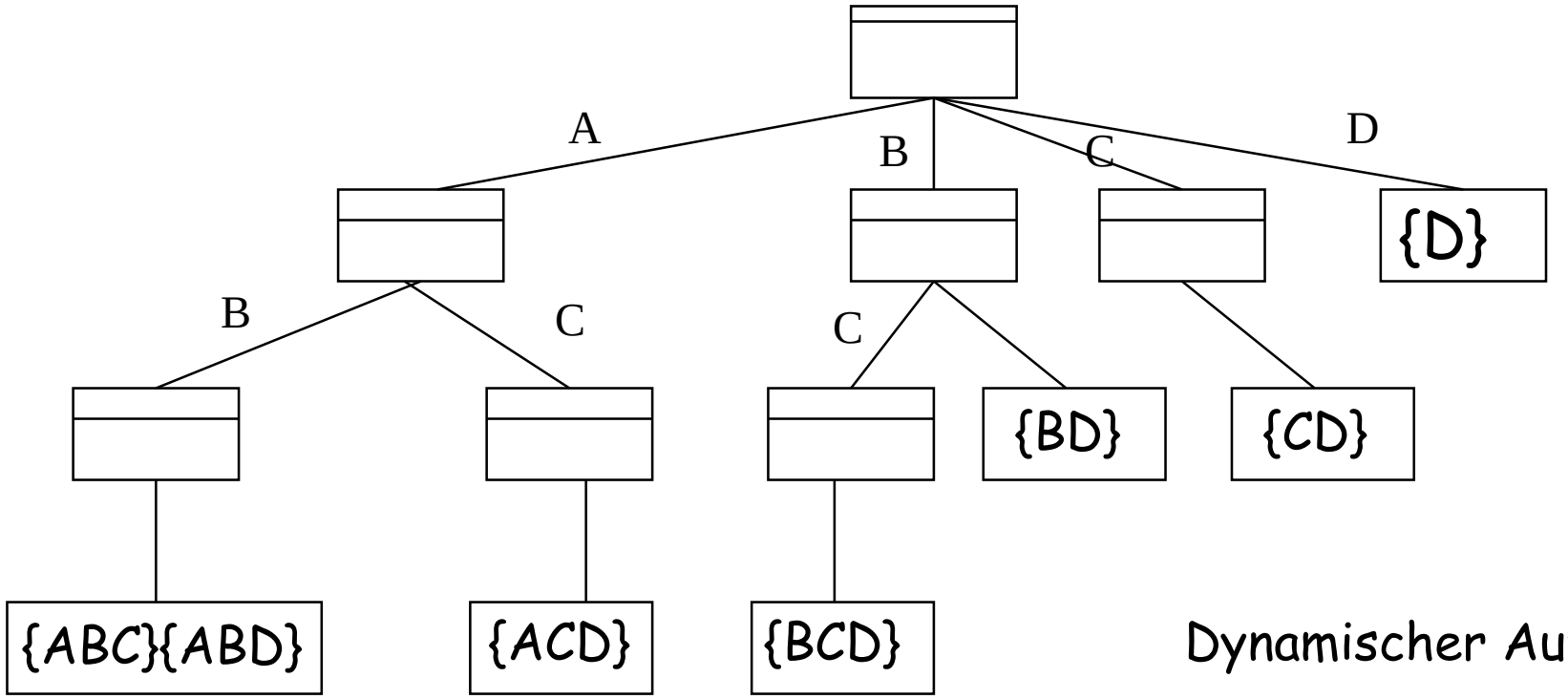
$l_1 = \{i_1, \ldots, i_{k-1}, i_k\}$ $c_1 = \{i_1, \ldots, i_{k-1}\} \rightarrow \{i_k\}$ $conf_1$
$l_1 = \{i_1, \ldots, i_{k-1}, i_k\}$ $c_2 = \{i_1, \ldots\} \rightarrow \{i_{k-1}, i_k\}$ $conf_2$
...
$l_1 = \{i_1, \ldots, i_{k-1}, i_k\}$ $c_k = \{i_1\} \rightarrow \{\ldots, i_{k-1}, i_k\}$ $conf_k$

$conf_1 \geq conf_2 \geq \ldots \geq conf_k$

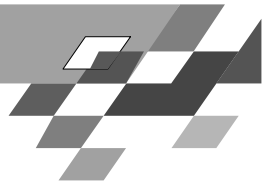


Implementierung

- Hash-Tree für den Präfixbaum, der sich aus der Ordnung der Elemente in den Mengen ergibt.
- An jedem Knoten werden Schlüssel und Häufigkeit gespeichert.

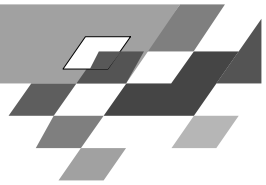


Dynamischer Aufbau



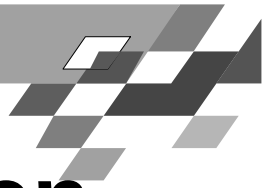
Was wissen Sie jetzt?

- 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.
- 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.



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 smin und confmin reicht bei der Warenkorbanalyse meist aus.
- Apriori liefert unglaublich viele Regeln.
- Die Regeln sind höchst redundant.
- Die Regeln sind irreführend, weil die Kriterien die apriori Wahrscheinlichkeit nicht berücksichtigen.
  Wenn sowieso alle Cornflakes essen, dann essen auch hinreichend viele Fußballer Cornflakes.



Prinzipien für Regelbewertungen

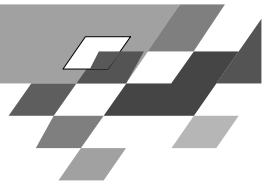
1. RI($A \rightarrow B$) = 0, wenn $|A \rightarrow B| = (|A| \cdot |B|) / |r|$
   A und B sind unabhängig.
2. RI($A \rightarrow B$) steigt monoton mit $|A \rightarrow B|$.
3. RI($A \rightarrow B$) fällt monoton mit $|A|$ oder $|B|$.

Also: RI > 0, wenn $|A \rightarrow B| > (|A| \cdot |B|) / |r|$
d.h., wenn A positiv mit B korreliert ist.

RI < 0, wenn $|A \rightarrow B| > (|A| \cdot |B|) / |r|$
d.h., wenn A negativ mit B korreliert ist.

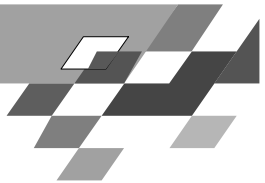
Wir wissen, dass immer $|A \rightarrow B| \leq |A| \leq |B|$ gilt, also
RImin wenn $|A \rightarrow B| = |A|$ oder $|A| = |B|$
RImax wenn $|A \rightarrow B| = |A| = |B|$

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
  aktiv-militär → kein-Dienst-in-Vietnam mit 90% Konfidenz.
  Tatsächlich ist s(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
  Basketball → Cornflakes hat Konfidenz 66%
  Tatsächlich senkt aber Basketball die Cornflakes Häufigkeit!



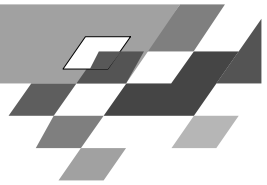
Signifikanztest

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

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

- Die Signifikanz der Korrelation zwischen A und B ist:

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



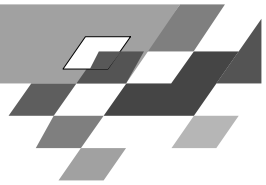
Sicherheitsmaß

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

- Wenn conf(A → B) > s(B)
  CF(A→B)= conf(A→B) – s(B)/(1-s(B))
- Wenn conf(A→B) < s(B)
  CF(A→B)= conf(A→B)
- Sonst
  CF(A→B)= 0.

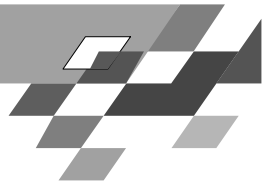
Das Sicherheitsmaß befolgt die Prinzipien für Regelbewertung.

Wendet man Signifikanztest oder Sicherheitsmaß an, erhält man weniger (irrelevante, irreführende) Assoziationsregeln.



Was wissen Sie jetzt?

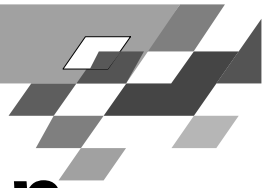
- Sie haben drei Prinzipien für die Regelbewertung kennengelernt:
  - Unabhängige Mengen sollen mit 0 bewertet werden.
  - Der Wert soll höher werden, wenn die Regel mehr Belege hat.
  - Der Wert soll niediger werden, wenn die Mengen weniger Belege haben.
- Sie haben drei Maße kennengelernt, die den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



Aktuelle Forschung

- Bessere Kriterien als support und Konfidenz
- Kondensierte Repräsentationen
- Anfrageoptimierung im Sinne induktiver Datenbanken durch constraints

- Die erste Verbesserung haben wir schon gesehen.
- Hier sehen wir die zweite Verbesserung.
- Die Konferenzen KDD, PKDD und ICDM sind aber voll von Beiträgen zu „frequent itemsets"!



Kondensierte Repräsentationen

Ersetzen der Datenbank bzw. der Baumstruktur durch eine kondensierte Repräsentation,

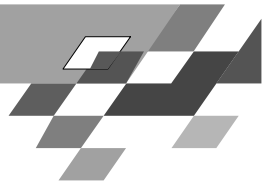
- die kleiner ist als die ursprüngliche Repräsentation und
- aus der wir alle häufigen Mengen und ihre Häufigkeit ableiten können, ohne noch mal die Daten selbst anzusehen.

Kondensierte Repräsentationen für Assoziationsregeln:

- Closed item sets
- Free sets

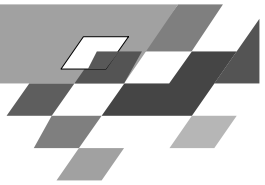
Operator, der die Menge aller Assoziationsregeln ableitet:

- Cover operator



Wir erinnern uns...

- Hypothesen werden in einem Verband angeordnet.
- Ein Versionenraum gibt die möglichen Hypothesen an, die zu den gegebenen Daten passen – durch weitere Daten wird der Versionenraum weiter eingeschränkt:
  - Wenn ein positives Beispiel nicht abgedeckt ist, wird die Menge der speziellsten Hypothesen generalisiert,
  - Wenn ein negatives Beispiel abgedeckt ist, wird die Menge der generellsten Hypothesen spezialisiert.



In anderen Worten:

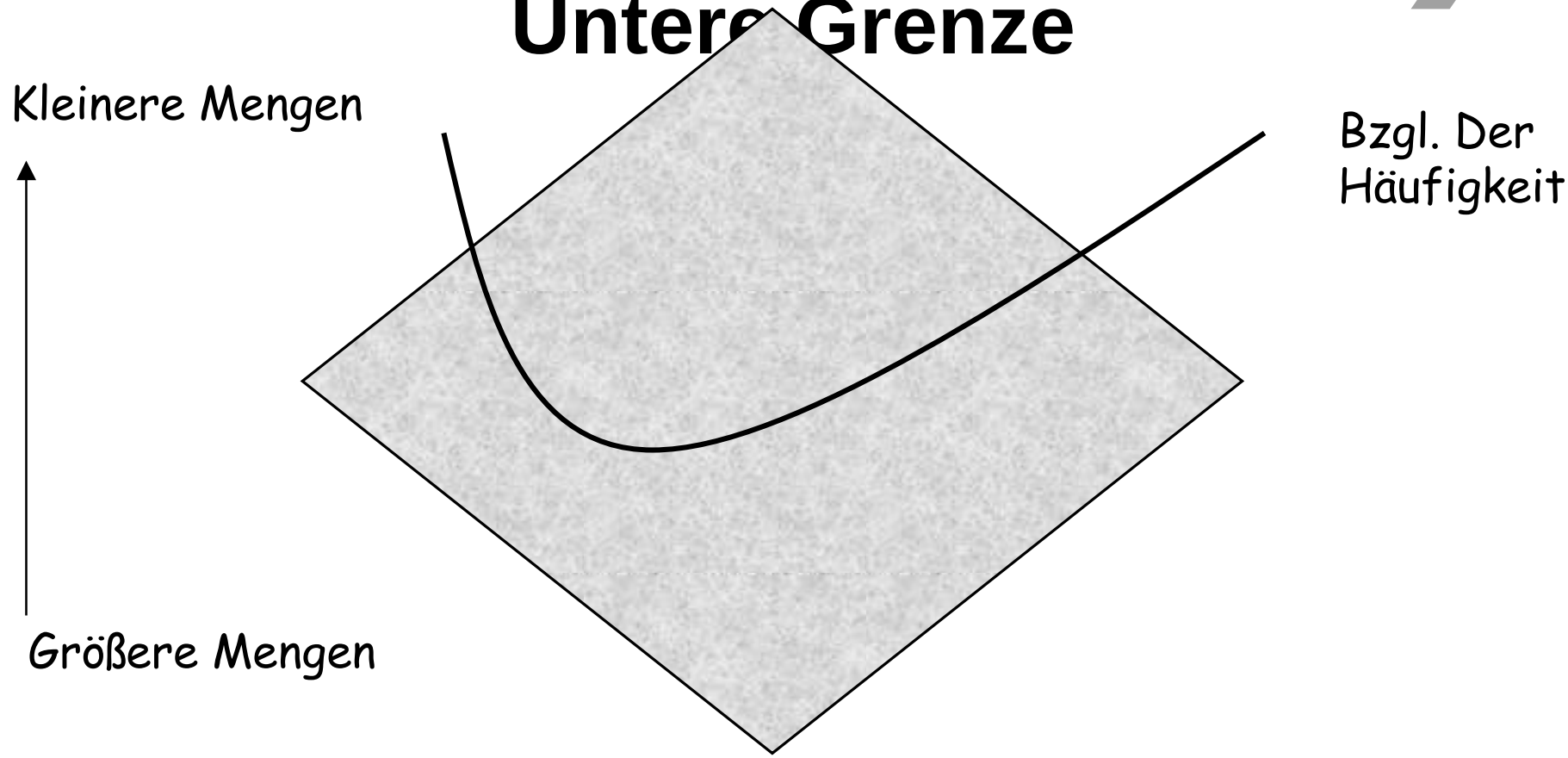
Wir hätten gern einen Versionenraum!
Der Versionenraum ist kleiner als der Hypothesenraum.
Außerhalb des Versionenraums kann das Lernziel nicht liegen.

Wir müssen also aus den Beispielen

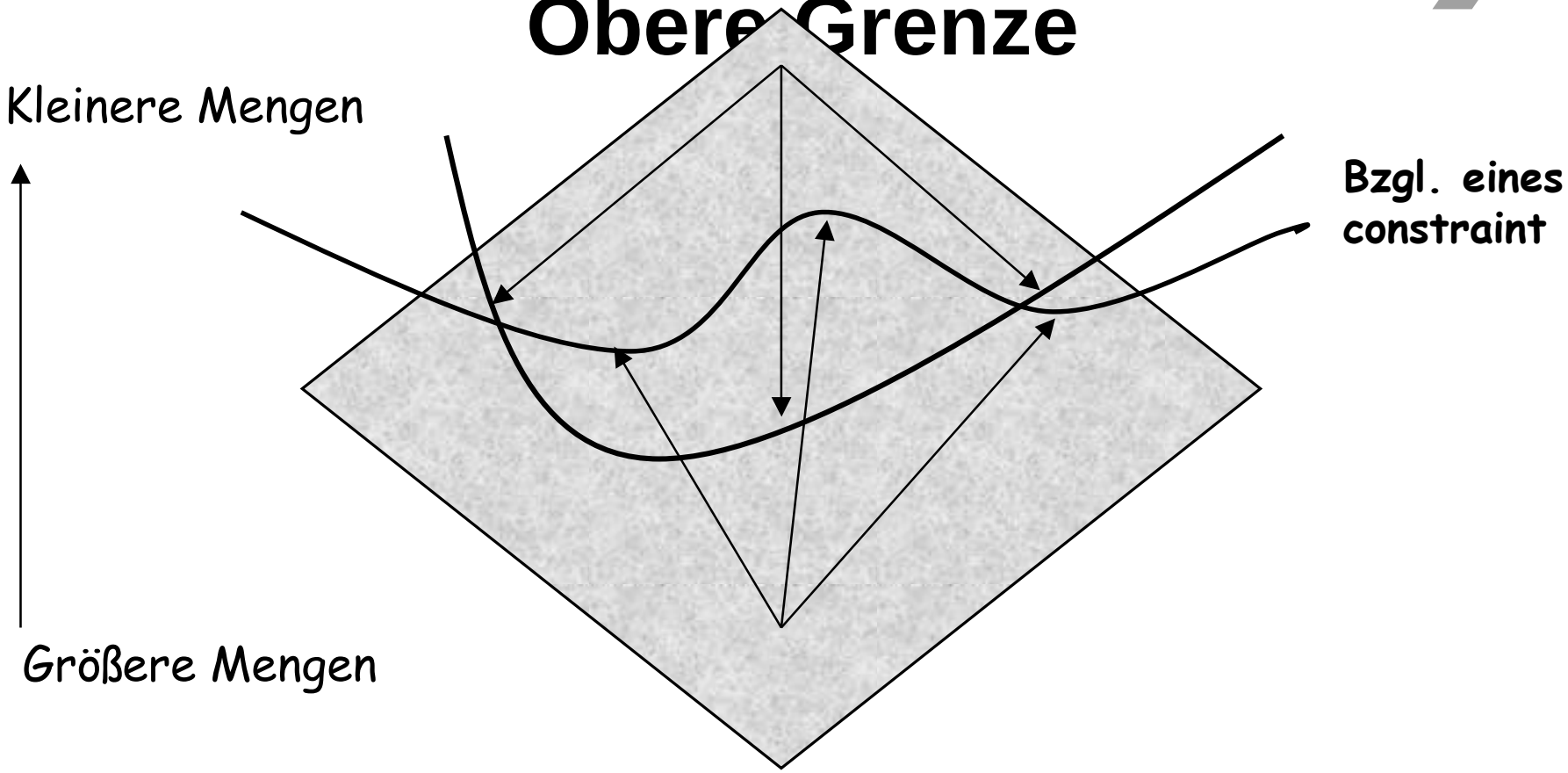
- eine untere Grenze und
- eine obere Genze konstruieren.

Eine Halbordnung bzgl. Teilmengenbeziehung haben wir schon.

Die Grenzen haben wir auch.
Gemerkt?



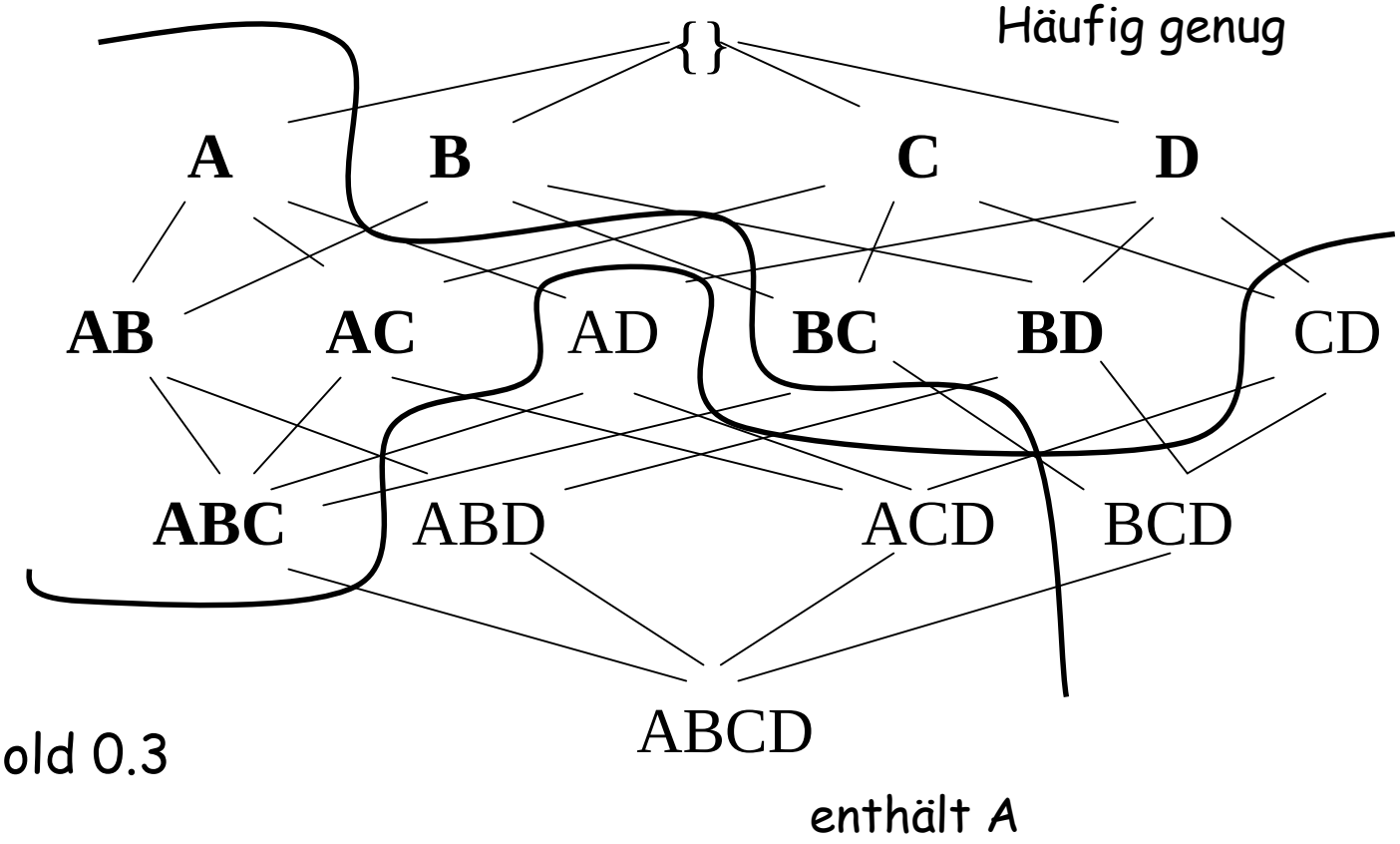
- Wenn eine Menge häufig ist, so auch all ihre Teilmengen. (Anti-Monotonie)
- Beschneiden der Ausgangsmengen für die Kandidatengenerierung gemäß dieser Grenze!



- Monotonie der Seltenheit: Wenn eine Teilmenge selten ist, so auch jede Menge, die sie enthält. Seltenheit ist ein constraint.
- Beschneidung der Kandidatengenerierung nach der Monotonie.

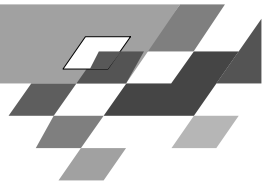
Beispiel

A	B	C	D
1	0	1	0
1	1	1	0
0	1	1	1
0	1	0	1
1	1	1	0



Frequency threshold 0.3

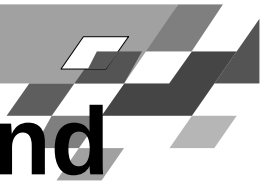
Dank an Jean-Francois Boulicaut!



Closed Item Sets

A	B	C	D
1	1	1	1
0	1	1	0
1	0	1	0
1	0	1	0
1	1	1	1
1	1	1	0

- closure(S) ist die maximale Obermenge (gemäß der Teilmengenbeziehung) von S, die noch genauso häufig wie S vorkommt.
- S ist ein *closed item set*, wenn closure(S)=S.
- Bei einem Schwellwert von 0,2 sind alle Transaktionen häufig genug.
- Closed sind: C, AC, BC, ABC, ABCD
  keine Obermenge von C kommt auch 6 mal vor;
  A kommt 5 mal vor, aber auch die Obermenge AC und keine Obermenge von AC
  ...



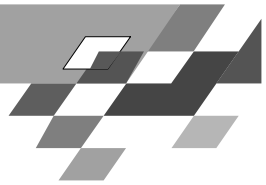
Kondensierte Repräsentation und Ableitung

Closed item sets sind eine kondensierte Repräsentation:

- Sie sind kompakt.
- Wenn man die häufigen closed item sets C berechnet hat, braucht man nicht mehr auf die Daten zuzugreifen und kann doch alle häufigen Mengen berechnen.

Ableitung:

- Für jede Menge S prüfen wir anhand von C:
  Ist S in einem Element X von C enthalten?
  - Nein, dann ist S nicht häufig.
  - Ja, dann ist die Häufigkeit von S ungefähr die von X.
    Wenn es in mehreren Elementen von C vorkommt, nimm die maximale Häufigkeit!



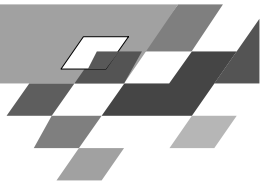
Freie Mengen (free sets)

- Eine Menge S ist frei, wenn es keine Regel mit Konfidenz=1 zwischen ihren Elementen gibt, d.h.

$$\neg\exists X, Y \left| S = X \cup Y, Y \neq \{\ \}, X \Rightarrow Y \right.$$

  - Eine Menge S ist δ-frei, wenn es keine Regel mit weniger als δ Ausnahmen zwischen ihren Elementen gibt.

  - Die closed sets sind die closure der freien Mengen!
    Man kann die closed sets aus den freien Mengen berechnen.
  - Freiheit ist eine anti-monotone Eigenschaft von Mengen.
    Deshalb kann man die freien Mengen effizient berechnen.

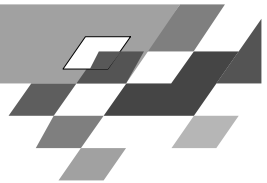


Beispiel

A	B	C	D
1	1	1	1
0	1	1	0
1	0	1	0
1	0	1	0
1	1	1	1
1	1	1	0
5	4	6	2

- Bei einem Schwellwert von 0,2 sind die häufigen freien Mengen:
  {}, A,B,D,AB
- Closed sind: C, AC, BC, ABC, ABCD
- Closure({})=C
  closure(A)=AC
  closure(B)= BC
  closure(D)=ABCD
  closure(AB)=ABC

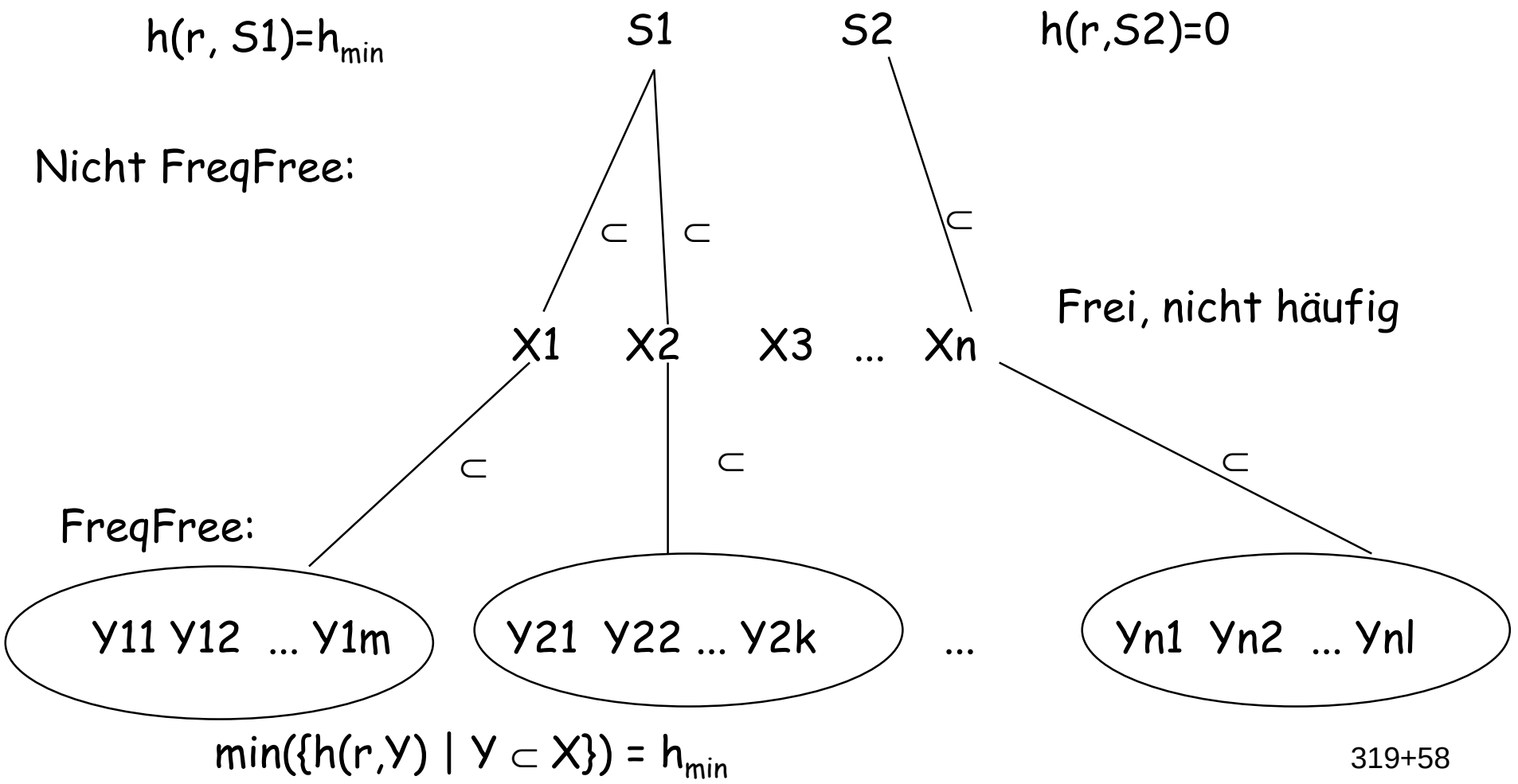
"Unfreie" Mengen: AD: D $\Rightarrow$ A, BD: D $\Rightarrow$ B, ABD
C:{} $\Rightarrow$ C, AC: A $\Rightarrow$ C, BC: B $\Rightarrow$ C, CD: D $\Rightarrow$ C, ABC, ADC, BCD, ABCD

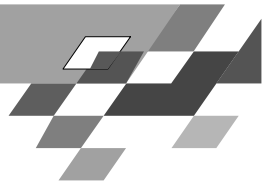


Arbeiten mit freien Mengen

- Free(r, δ): Eine Menge X ist δ-frei, wenn es in r keine Regel zwischen ihren Elementen mit weniger als δ Ausnahmen gibt.
- Freq(r, σ): $\{X \mid X \subseteq R, |X \in r| / |r| \geq \sigma\}$
- FreqFree(r, σ, δ): Freq (r, σ) $\cap$ Free(r, δ)
- Negative Grenze Bd-(r, σ, δ): $\{X \mid X \subseteq R, X \notin$ FreqFree(r, σ, δ) und $\forall Y \subset X, Y \in$ FreqFree (r, σ, δ) $\}$
  Also die kürzesten Mengen, die gerade nicht häufig und frei sind, deren Teilmengen aber häufig und frei sind.
- Wir schätzen die Häufigkeit einer Menge S so ab:
  $\exists X \subseteq S$ und X ist δ-frei, aber nicht σ-häufig, dann
  nimm 0 als Häufigkeit von S.
  Sonst nimm die kleinste Anzahl im Vorkommen der Teilmengen X als Häufigkeit von S.

Abschätzung

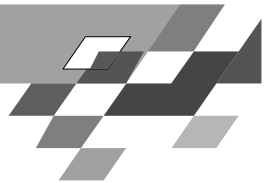




MinEx

- Statt alle häufigen Mengen zu suchen, brauchen wir nur noch alle FreqFree(r, σ, δ) zu suchen.
- Bottom-up Suche im Halbverband der Mengen
  beginnt beim leeren Element, nimmt dann alle 1-elementigen Mengen,...
  endet bei den größten Mengen, die noch FreqFree(r, σ, δ) sind.
- Der Test, ob Mengen frei sind, erfordert das Bilden von strengen Regeln und erlaubt das Pruning der Mengen, in denen solche gefunden wurden.

Algorithmus von Jean-Francois Boulicaut

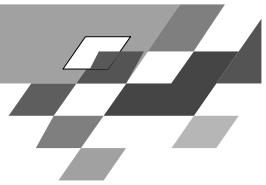


Algorithmus (abstrakt)

Gegeben eine binäre Datenbasis r über Objekten R und
die Schwellwerte σ und δ,

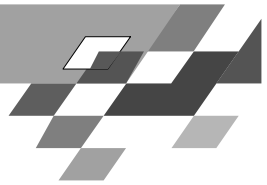
Gebe FreqFree(r, σ, δ) aus.

1. C_0:={ {} }
2. i:=0
3. **While** $C_i \neq \{\}$ **do**
4. FreqFree $_i$:= {X |X $\in$ C_i, X ist σ-häufig und δ-frei}
5. C_{i+1}:= {X | X $\subseteq$ R, $\forall$Y $\subset$ X, Y $\in$ FreqFree$_j$ (r, σ, δ), j $\leq$ i }\
$\cup_{j \leq i} C_j$
6. i:=i+1 **od**
7. **Output** $\cup_{j < i}$ FreqFree $_j$



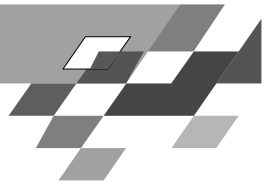
Pruning

- In der i-ten Iteration werden die δ–starken Regeln der Form X → {A} berechnet, wobei
  X häufig und frei ist auf der i-ten Ebene und
  $A \subseteq R \backslash X$.
- Das Ergebnis wird verwendet, um alle nicht δ -freien Mengen zu entfernen – sie sind keine Kandiaten mehr in der i+1-ten Iteration.



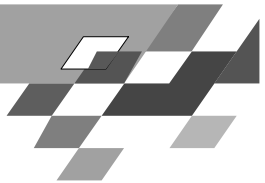
Eigenschaften von MinEx

- Der Algorithmus ist immer noch aufwändig, aber schneller als APRIORI und schneller als die Verwendung von closed sets.
- Der Algorithmus ist exponentiell in der Menge .
- Der Algorithmus ist linear in der Menge der Datenbanktupel, wenn δ im selben Maße steigt wie die Zahl der Tupel.
  Wir verdoppeln δ, wenn wir die Tupelzahl verdoppeln.
- Der Algorithmus approximiert das „wahre" Ergebnis.
  In der Praxis ist eine Abweichung von 0,3% aber kein Problem.

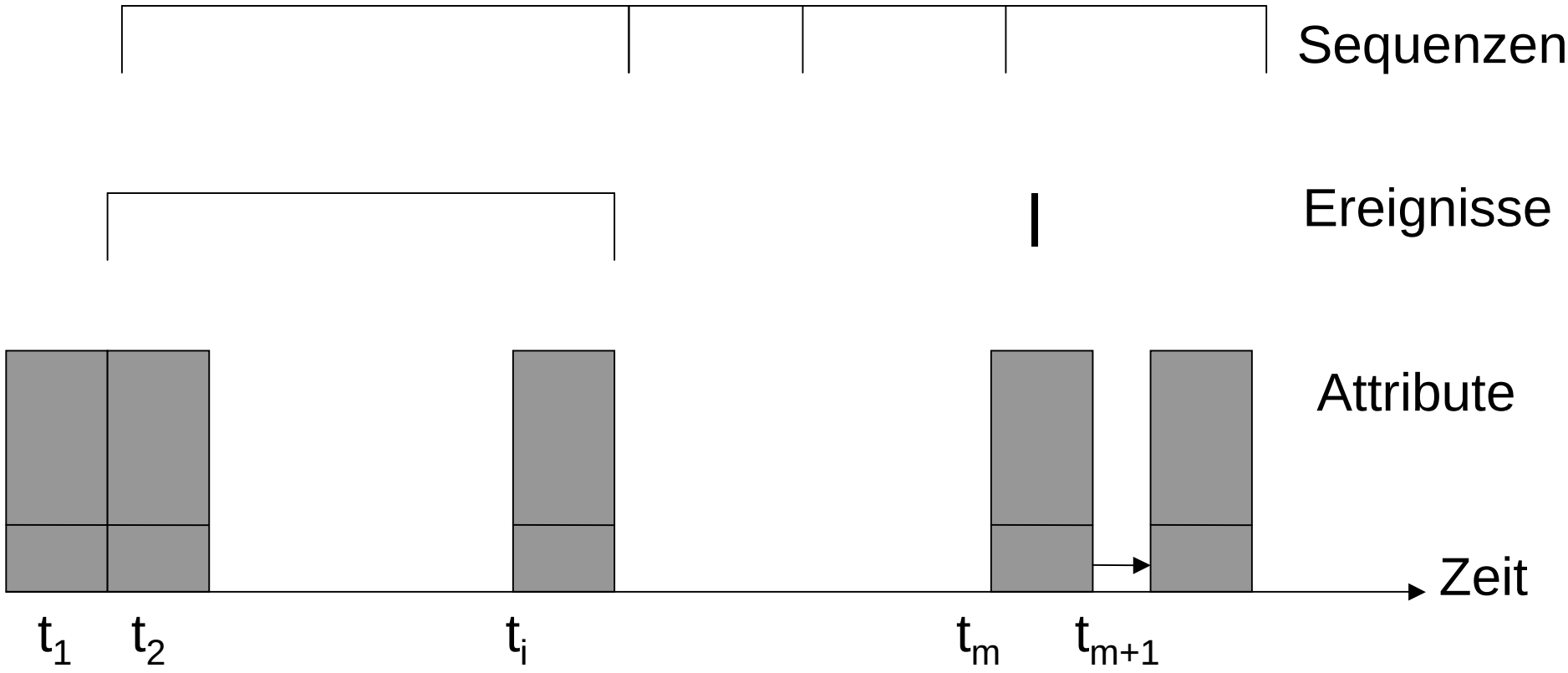


Was wissen Sie jetzt?

- Es gibt zwei Repräsentationen, die weniger Elemente für eine Suche nach häufigen Mengen ausgeben als eben alle häufigen Mengen. Aus diesen Repräsentationen können alle häufigen Mengen hergeleitet werden.
  - Die closed sets sind maximale Obermengen von S mit derselben Häufigkeit wie S.
  - Die free sets sind Mengen, aus denen man keine Assoziationsregeln machen kann.
- Wenn man die häufigen freien Mengen berechnet, hat man die untere Grenze im Versionenraum für Assoziationsregeln gefunden.
- Der Algorithmus MinEx findet diese Grenze.

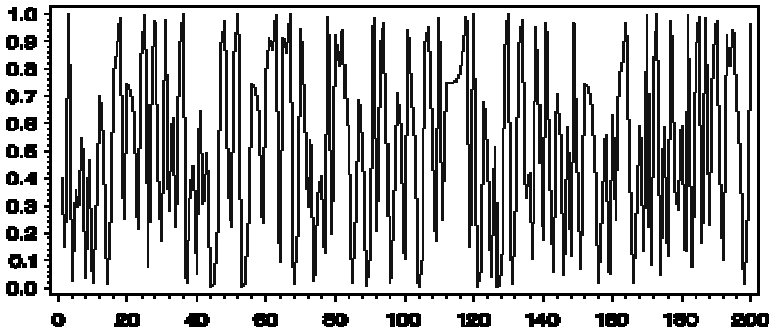


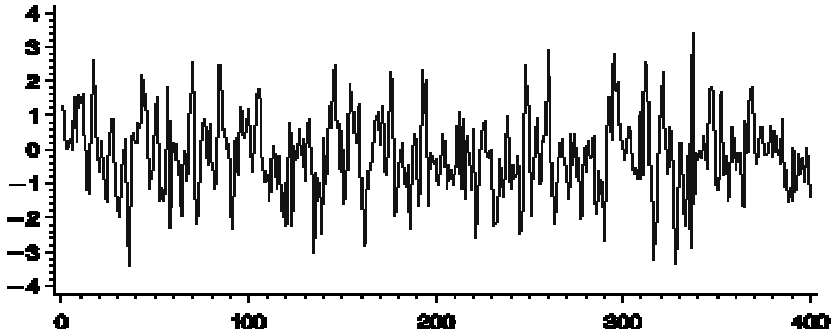
Zeitphänomene



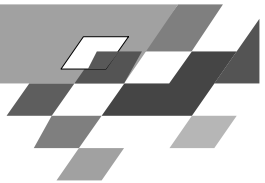
Beispiele für Zeitreihen

- Messwerte von einem Prozess
  - Intensivmedizin
  - Aktienkurse
  - Wetterdaten
  - Roboter





Kontinuierliche Messung in z.B. Tagen, Stunden, Minuten, Sekunden

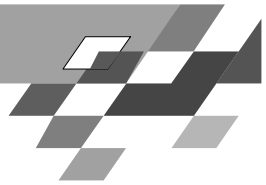


Beispiele für Ereignisse

- Datenbankrelationen
  - Vertragsdaten, Verkaufsdaten, Benutzerdaten
  - Lebenssituation (Einkommen, Alter)

Verkäufe	Monat	Anzahl	Verkäufer	...
	Juni	256	Meier	...
	...	...	...	...

Ereignisse mit Zeitangaben in Jahren, Monaten, Tagen

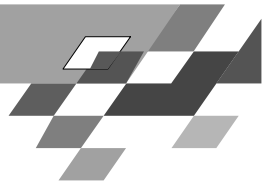


Granularität

- Eine Granularität ist eine Abbildung von natürlichen Zahlen (oder Zeichenketten) auf Teilmengen der Zeitwerte, so dass gilt:
  1. Wenn i < j und G(i), G(j) nicht leer, dann ist jedes Element von G(i) kleiner als alle Elemente von G(j).
  2. Wenn i < k < j und G(i) und G(j) nicht leer, dann ist G(k) auch nicht leer.

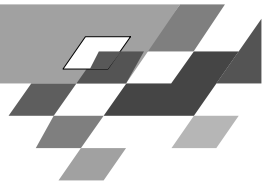
1. Der Index i,k,j bezeichnet eine Kodierung der Zeiteinheiten. Die Zeiteinheiten überlappen sich nicht.
2. Die Teilmengen von Indizes folgen aufeinander. Tage, Arbeitstage, Wochen, Semester, Kalenderjahre sind Zeiteinheiten.

- Beispiel: Jahre seit 2000 sei definiert als G mit G(i)={} für i<1, G(1)=alle Zeit im Jahre 2000, G(i+1)= alle Zeit in 2001,...



Temporale Module

- Temporales Modulschema (R,G), wobei R ein Relationenschema ist und G eine Zeitgranularität.
- Temporales Modul (R,G,p), wobei p die Zeitfensterabbildung von natürlichen Zahlen auf Tupel in der Zeiteinheit ist.
- Zu einer Zeiteinheit G(i) liefert p alle Tupel, die in der entsprechenden Zeit gelten.
- Beispiel: Sei in R das Jahresgehalt für Mitarbeiter und sei G Jahre seit 2000, dann liefert p für i=1 alle Gehälter im Jahre 2000.

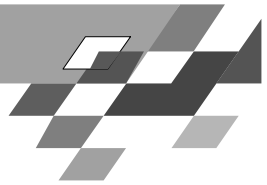


Temporale Datenbank

- Das Schema einer temporalen Datenbank ist eine Menge von temporalen Modulschemata.
- Eine Menge von temporalen Modulen bildet eine temporale Datenbank.

Claudio Bettini, Sushil Jajodia, Sean X. Wang (1998)

„Time Granularities in Databases, Data Mining, and Temporal Reasoning"
Springer



Beispiel

t1(kurs)=CS50=t2(kurs)

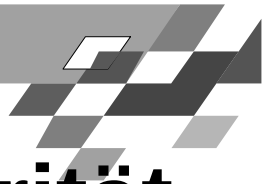
p(1993-5-26)=[CS50, 3, Woo, 2 000, 50]

p(1993-5-30)=[CS50, 3, Woo, 2 000, 45] ...

G sei Tag als Einheit, G(1)= 1993-5-26, G(2)= 1993-5-30

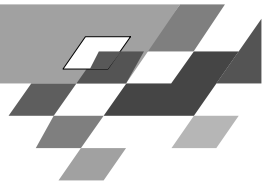
H sei Kalenderwoche als Einheit H(22)={1993-5-26, 1993-5-27, 1993-5-28, 1993-5-29, 1993-5-30, 1993-5-31, 1993-6-1}

kurs	credits	wimi	gehalt	#studis	tag
CS50	3	Woo	2 000	50	1993-5-26
CS50	3	Woo	2 000	45	1993-5-30
CS50	3	Woo	2 500	48	1993-6-2
CS50	3	Lee	2 000	46	1993-6-13
CS50	3	Lee	2 000	44	1993-6-16
CS50	3	Lee	2 000	43	1993-6-20



Partielle Ordnungen der Granularität

- Wann ist eine Granularität feiner als eine andere?
  Z.B. Tag, Woche
- Wann ist eine Granularität eine Untergranularität einer anderen?
  Wenn es für jedes G(i) einen Index j gibt, so dass G(i)=H(j), dann ist G Untergranularität von H.
- Wann deckt eine Granularität eine andere ab?
  Wenn der Bildbereich von G im Bildbereich von H enthalten ist, dann deckt wird G von H abgedeckt.

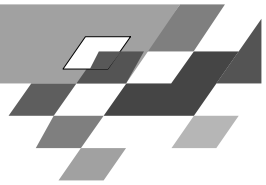


Schemaentwurf

1. Die Attribute tag und kurs sollen Schlüssel sein.
2. Die funktionale Abhängigkeit kurs → credits soll gegeben sein.
3. Das Gehalt eines Mitarbeiters ändert sich nicht innerhalb eines Monats.
4. Mitarbeiter wechseln sich nicht innerhalb derselben Woche ab.

kurs	credits	wimi	gehalt	#studis	tag
CS50	3	Woo	2 000	50	1993-3-3
CS50	3	Woo	2 000	45	1993-3-8
CS50	3	Woo	2 500	48	1993-4-5
CS50	3	Lee	2 000	46	1993-4-10
CS50	3	Lee	2 000	44	1993-5-7
CS50	3	Lee	2 000	43	1993-5-12

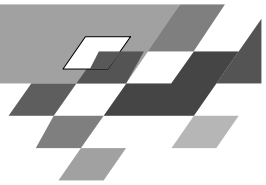
Oh!



Anomalien

- Redundanz: credits, gehalt
- Einfügeanomalie: Woos Gehalt in einem Tupel ändern und in den anderen desselben Monats lassen...
- Löschanomalie: Wenn der Kurs gelöscht wird, verlieren wir die Mitarbeiternamen...

kurs	credits	wimi	gehalt	#studis	tag
CS50	3	Woo	2 000	50	1993-5-26
CS50	3	Woo	2 000	45	1993-5-30
CS50	3	Woo	2 500	48	1993-6-2
CS50	3	Lee	2 000	46	1993-6-13
CS50	3	Lee	2 000	44	1993-6-16
CS50	3	Lee	2 000	43	1993-6-20



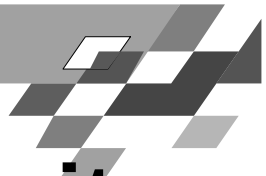
Dekomposition

wimi	gehalt	monat
Woo	2 000	1993-5
Woo	2 500	1993-6
Lee	2000	1993-6

kurs	credits
CS50	3

kurs	wimi	kalender woche
CS50	Woo	22
CS50	Woo	23
CS50	Lee	24
CS50	Lee	25

kurs	#studis	tag
CS50	50	1993-5-26
CS50	45	1993-5-30
CS50	48	1993-6-2
CS50	46	1993-6-13
CS50	44	1993-6-16
CS50	43	1993-6-20

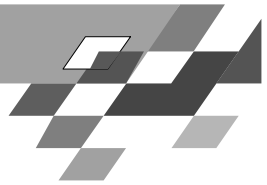


Temporale funktionale Abhängigkeiten

- Sei (R, G, p) ein temporales Modul, dann gilt $X \rightarrow_H Y$ gdw. wenn gilt
  1. t1(X) = t2(X)
  2. t1 in p(i) und t2 in p(j)
  3. Es gibt ein z mit $G(i) \cup G(j) = G(i,j)$ und $G(i,j) \subseteq H(z)$

  dann t1(Y)=t2(Y).

kurs $\rightarrow_{kalenderwoche}$ wimi
kurs $\rightarrow_{tag}$ #studis
wimi $\rightarrow_{monat}$ gehalt



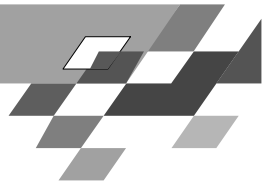
Beispiel1

kurs $\rightarrow_{kalenderwoche}$ wimi gilt, denn wenn

1. t1(kurs)=CS50=t2(kurs),
2. t1 in p(datum1) und t2 in p(datum5)
3. G(i)=datum1, G(j)=datum5, {datum1, datum5} in H(z)={datum1, datum2, datum3, datum4,datum5, datum6, datum7}

Dann t1(wimi)=t2(wimi)

kurs	credits	wimi	gehalt	#studis	tag
CS50	3	Woo	2 000	50	1993-5-26
CS50	3	Woo	2 000	45	1993-5-30
CS50	3	Woo	2 500	48	1993-6-2
CS50	3	Lee	2 000	46	1993-6-13
CS50	3	Lee	2 000	44	1993-6-16
CS50	3	Lee	2 000	43	1993-6-20



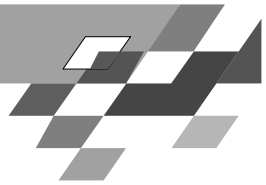
Beispiel2

kurs $\rightarrow_{monat}$ #studis gilt nicht, denn

1. t1(kurs)=CS50=t2(kurs),
2. t1 in p(1993-5-26) und t2 in p(1993-5-30)
3. G(i)= 1993-5-26, G(j)= 1993-5-30, G(i,j) in H(mai)

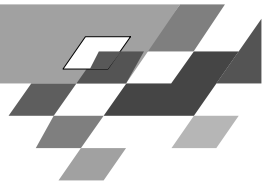
aber t1(#studis)=50 und t2(#studis)=45

kurs	credits	wimi	gehalt	#studis	tag
CS50	3	Woo	2 000	50	1993-5-26
CS50	3	Woo	2 000	45	1993-5-30
CS50	3	Woo	2 500	48	1993-6-2
CS50	3	Lee	2 000	46	1993-6-13
CS50	3	Lee	2 000	44	1993-6-16
CS50	3	Lee	2 000	43	1993-6-20



Temporaler Oberschlüssel

- Eine Menge von Attributen X heißt temporaler Oberschlüssel eines Moduls (R, G), wenn $X \rightarrow_G R$ logisch aus der Menge der temporalen funktionalen Abhängigkeiten folgt.
- $X \rightarrow_G Y$ folgt logisch aus TFD, wenn für jedes Modul, in dem alle Abhängigkeiten in TFD gelten, auch $X \rightarrow_G Y$ gilt.
- Wenn zwei Tupel zu (R,G) in derselben Zeiteinheit von G in den Attributen X dieselben Werte haben, dann sind sie insgesamt gleich.
- Trivialerweise ist stets R ein Oberschlüssel zu (R,G).



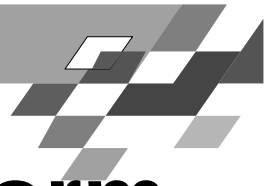
Temporale Projektion

- Sei M=(R,G,p) und $X \subseteq R$.
  πX (m) ist die Projektion auf (X, G, p1), wobei für alle i
  p1(i) = πX (p(i))
  mit der üblichen Projektion πX.

Für alle Schnappschüsse i werden die Tupel in m auf die Attribute X projiziert. Das Ergebnis ist $m' = \bigcup_i \pi\ X$

- Sei F die Menge der temporalen funktionalen Abhängigkeiten und Z eine Menge von Attributen, dann ist $\pi Z\ (F) = \{X \rightarrow_H Y \mid F \Rightarrow X \rightarrow_H Y, XY \subseteq Z\}$.

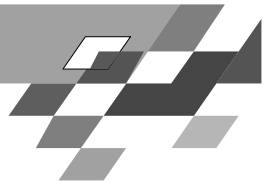
Man hat die temporalen funktionalen Abhängigkeiten mit den Attributen in Z.



Temporale Boyce-Codd Normalform

- Sei M=(R,G) ein temporales Modulschema mit F als Menge der temporalen funktionalen Abhängigkeiten.
  M ist in temporaler BCNF, wenn für jede temporale funktionale Abhängigkeit $X \rightarrow_H Y$, die aus F logisch folgt (wobei $X, Y \subseteq R$, $Y \not\subset X$, mindestens eine Zeiteinheit von G wird von einer in H abgedeckt) gilt:
  1. $X \rightarrow_G R$ folgt logisch aus F, dh. X ist ein temporaler Oberschlüssel
  2. Für alle $i \neq j$ von G gilt <u>nicht</u>: $X \rightarrow Y \in \pi G(i,j)$ (F).

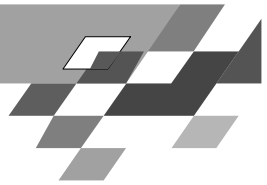
1. ist die temporale Version der üblichen Oberschlüsselbedingung.
2. verhindert, dass es temporale funktionale Abhängigkeiten mit H gibt, wobei zwei Zeiteinheiten von G durch eine Zeiteinheit von H abgedeckt werden.



Beispiel

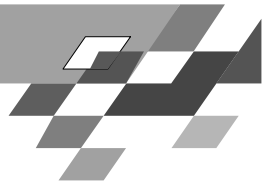
- M=(Kurse, tag, p)
- F:{kurs→credits, wimi→$_{monat}$ gehalt, kurs→$_{woche}$ wimi, kurs→$_{tag}$ #studis}
- F⇒ kurs→ $_{woche}$ wimi (wobei kurs, wimi ⊆ Kurse, wimi ⊄ kurs, ein Tag wird von einer Woche abgedeckt)
- Es soll gelten:
  1. kurs →$_{tag}$ credits, wimi, gehalt, #studis -- stimmt
  2. Die temporale Relation auf alle Paare von Tagen projiziert, gibt es dort nicht die funktionale Abhängigkeit kurs→wimi -- stimmt nicht!

  Es gibt zwei Tage derselben Woche, so dass dort kurs→wimi gilt.



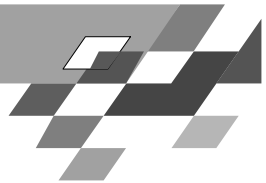
Was wissen wir jetzt?

- Wir haben die Zeit aus einer Datenbankperspektive gesehen.
- Normalerweise wird ein Zeitattribut in einer Datenbank gar nicht anders als andere Attribute behandelt.
- Das kann aber zu irreführenden oder redundanten Schemata führen, wenn wir eigentlich mehrere Granularitäten der Zeit haben.
- Deshalb arbeitet der Bereich der temporalen Datenbanken daran, alle Formalisierungen der Datenbanken auf eine besondere Berücksichtigung der Zeit hin zu erweitern.
- Gesehen haben wir funktionale Abhängigkeiten, Projektion und Normalform.



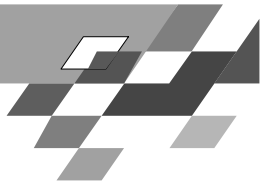
Zum Behalten

- Selbst bei normalen Datenbanken sollte man bei Zeitstempeln aufpassen:
  - Gibt es unterschiedliche Granularitäten? Tag, Woche, Monat
  - Besser ist nur eine Granularität je Tabelle, für verschiedene Granularitäten besser verschiedene Tabellen anlegen!
- Wenn unterschiedliche Granularität vorhanden ist:
  - Welche Attribute sind bei welcher Zeiteineit veränderlich?
  - Wenn Attribute bei einer Zeiteinheit nicht verändert werden können, sollen sie auch nicht mit dieser gestempelt werden!
  - Attribute sollen nur mit der Granularität aufgeführt werden, bei der sich ihre Werte ändern!



Lernaufgaben für Ereignisse

- Wie finde ich Ereignisse in Zeitreihen?
- Wie finde ich Episoden (häufige Mengen von Ereignissen in partieller Ordnung) in Ereignissequenzen?
  Wie will ich die Zeit in den Sequenzen darstellen:
  - Absolute Dauer
  - Zeit zwischen Prämisse und Konklusion
  - Relation zwischen Zeitintervallen (vor, während, nach...)

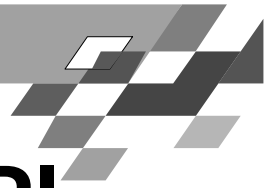


Lernaufgaben

Lernaufgaben bei einer gegebenen Sequenz von Ereignissen:

(Menge von Ereignissen in partieller Ordnung)

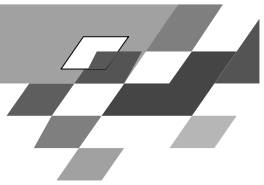
1. Finde häufige Episoden in Sequenzen [Mannila et al.]
   - Wenn A auftritt, dann tritt B in der Zeit T auf [Das et al.]
2. Beziehungen zwischen Zeit-Intervallen lernen [Höppner]
   - A startet vor B, B und C sind gleichzeitig, C und D überlappen sich, D endet genau, wenn E anfängt ...



Heikki Mannilas Ansatz:WINEPI

- E sind Attribute, genannt Ereignistypen.
- Ein Ereignis e ist ein Paar (A, t), wobei A in E und t integer.
- Eine Beobachtungssequenz s ist ein Zeitraum von Ts bis Te mit einer Folge s, die aus Ereignissen besteht:
  s=(<$(A_1, t_1), (A_2, t_2), ..., (A_n, t_n)$>, Ts, Te) wobei $t_i \leq t_{i+1}$
  und $Ts \leq t_i < Te$ für alle i=1...n
- Es geht darum, häufige Episoden in Sequenzen zu finden.
  Analog zu APRIORI.
- Anwendungen aus der Telekommunikation: Einbruchsversuche in ein Netzwerk, häufige Klickfolgen bei einer Web site, Nutzungsprofile,...

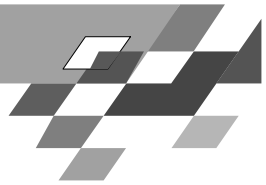
Heikki Mannila, Hannu Toivonen, Inkeri Verkamo "Discovery of frequent episodes in event sequences", Tech. Report C-1997-15 Univ. Helsinki



Fenster

- Ein Fenster w der Breite win ist ein Tripel (w, ts, te) und enthält die Ereignisse (A, t), bei denen ts ≤ t < te und ts ≤ Te und te > Ts. ACHTUNG, kein Tippfehler! Randereignisse werden so richtig gezählt, sonst kämen sie in weniger Fenstern vor als Ereignisse in der Mitte der Folge.

- Die Menge aller Fenster W(s,win) hat die Kardinalität Te-Ts + win-1.

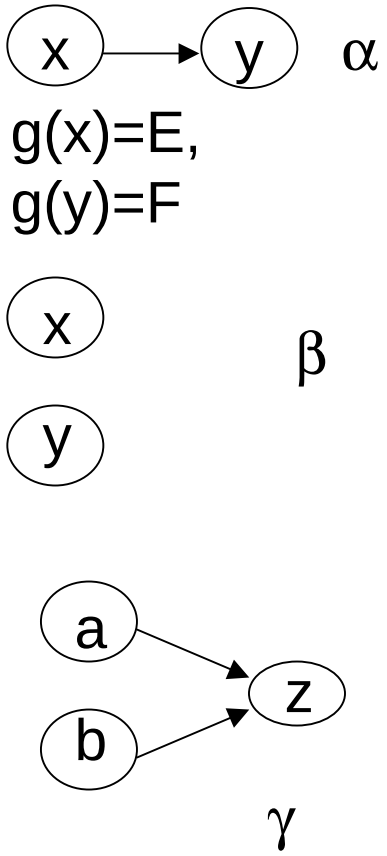


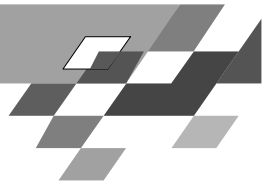
Beispiel

- **s**=(s, 29, 68)
  s=<(E,31), (D, 32), (F,33), (A,35), (B, 37), (C,38),(E,39),(F,40),...,(D,67)>
- Fensterbreite 5 ergibt z.B. die Folge:
  (<(A,35), (B, 37), (C,38),(E,39)>, 35,40)
  4 Ereignisse kommen in den 5 Zeitpunkten vor
  Das Ereignis, das an Zeitpunkt 40 vorkommt, ist nicht im Fenster
  (s, 35,40), sondern erst in dem (s, 36, 41).
- Das erste Fenster ist ({},25, 30) und das letzte ist (<(D,67)>,67,72).
- (D,67) kommt in 5 Fenstern der Breite 5 vor.
  Genauso oft wie etwa (B,37).
- Es gibt 68-29+5-1= 43 Fenster.

Episoden

- $\alpha=(V,\leq, g)$ ist eine serielle Episode, wenn für alle x, y in V gilt: $x \leq y$ oder $y \leq x$. V ist eine Menge von Knoten. g: $V \rightarrow E$.
- $\beta=(V, \leq, g)$ ist eine parallele Episode, wenn die Ordnungsrelation trivial ist (gilt nie).
- $\beta=(V, \leq, g) \angle \gamma=(V', \leq', g')$, wenn es eine eindeutige Abbildung f gibt, $f: V \rightarrow V'$ so dass $g(v)=g'(f(v))$ für alle v in V und für alle v,w in V mit $v \leq w$ gilt $f(v) \leq' f(w)$.
- Beispiel: β ist eine Unterepisode von γ, weil $f(x)=a$, $f(y)=b$ $\leq$ ist egal.





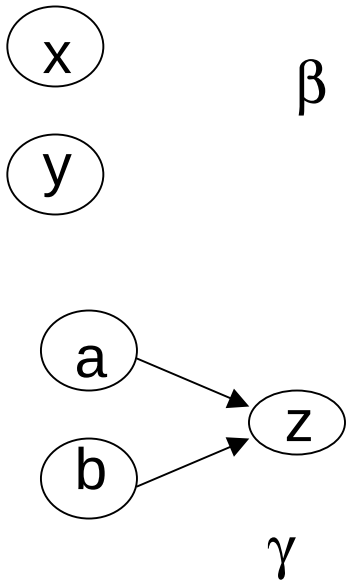
Episode ist in Folge

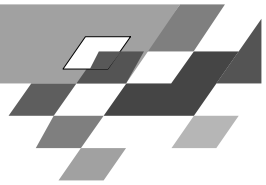
- Eine Episode $\alpha=(V,\leq, g)$ ist in einer Folge (occurs in) **s**=($\langle(A_1, t_1), (A_2, t_2), \ldots, (A_n, t_n)\rangle$, Ts, Te), wenn
  - Es gibt eine eindeutige Abbildung h:V → {1,…,n} so dass $g(x)= A_{h(x)}$ für alle x in V und
  - Für alle x,y ∈ V mit x ≠y und x ≤ y gilt: $t_{h(x)} \leq t_{h(y)}$

Beispiel

s=(<(*A*,35), (B, 37), (*C*,38),(E,39)>, 35,40)

- Mit *g*(x)=*A*, *g*(y)=B und
  h(x)=1, h(y)=2 ist β in s.
  Es gibt mehrere Abbildungen, so dass β in s ist, weil die Ordnung trivial ist.
- Mit *g*(a)=*A*, *g*(b)=B, *g*(z)=*C* und
  h(a)=1, h(b)=2, h(z)=3 ist γ in s
  $t_{h(a)} \leq t_{h(z)}$ und $t_{h(b)} \leq t_{h(z)}$



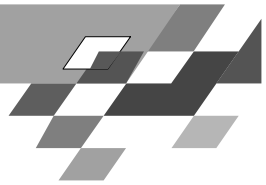


Häufigkeit einer Episode

- Die Häufigkeit einer Episode α in einer Folge s bei einer Fensterbreite win ist

$$fr(\alpha, s, win) = \frac{\left|\left\{w \in W(s, win) \middle| \alpha \text{ ist in } w\right\}\right|}{\left|W(s, win)\right|}$$

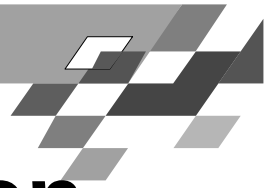
- Wir setzen einen Schwellwert min_fr, so dass α nur häufig ist, wenn fr(α,s,win)≥min_fr.
- Die Menge der häufigen Episoden wird geschrieben als $\mathcal{F}$(s,win,min_fr).



WINEPI: Regeln generieren

- Gegeben eine Menge E von Ereignistypen, eine Ereignisfolge s über E, eine Klasse $\mathcal{E}$ von Episoden, eine Fensterbreite win, ein Schwellwert min_fr und einer min_conf
- Finde Episodenregeln.

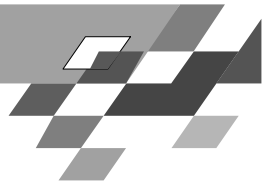
1. Berechne $\mathcal{F}$(s, win, min_fr); /* Finde häufige Episoden */
2. For all α in $\mathcal{F}$(s, win, min_fr) do /* Generiere Regeln */
3. for all $\beta \angle \alpha$ do
4. if $fr(\alpha)/fr(\beta) \geq$ min_conf then
5. gib aus $\beta \rightarrow \alpha$ mit conf=$fr(\alpha)/fr(\beta)$;



WINEPI: Finde häufige Episoden

- Gegeben eine Menge E von Ereignistypen, eine Ereignisfolge s über E, eine Klasse $\mathcal{E}$ von Episoden, eine Fensterbreite win und ein Schwellwert min_fr
- Finde die Menge häufiger Episoden $\mathcal{F}$(s,win,min_fr).

1. C_1:={$\alpha \in \mathcal{E}$ | | α |=1 }; /*Erste Kandidaten */
2. ℓ:= 1;
3. While $C_\ell \neq$ { } do
4. $\mathcal{F}_\ell$:={$\alpha \in C_\ell$ | fr(α, s, win) ≥min_fr}; /*Datenbankdurchlauf*/
5. ℓ:= ℓ +1;
6. C_ℓ:={$\alpha \in \mathcal{E}$ | | α |=ℓ und für alle $\beta \in \mathcal{E}$ mit $\beta \angle \alpha$, | β | < ℓ gilt $\beta \in \mathcal{F}_{|\beta|}$}; /*Kandidatengenerierung*/
7. For all ℓ do $\mathcal{F}_\ell$ ausgeben;



Repräsentation

- Episode als Vektor
  - sortiert lexikografisch (parallele Episoden) oder
  - sortiert nach ≤ (serielle Episoden)

  α= A A B C wird geschrieben: $\alpha[1]$=A $\alpha[2]$=A $\alpha[3]$=B $\alpha[4]$=C
- Sortierter Array für die Menge der Episoden

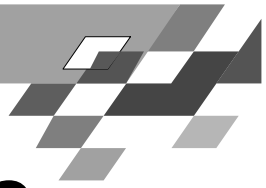
  $\mathcal{F}_\ell[1]$ erste Episode der Länge ℓ
  - sortiert nach gemeinsamen Unterepisoden der Länge ℓ-1

  $\mathcal{F}_4$: [1] A A B C

  [2] A A B D

  [3] A A B F
  - D.h.:Wenn $\mathcal{F}_\ell[i]$ und $\mathcal{F}_\ell[j]$ in den ersten ℓ-1 Ereignissen übereinstimmen, dann auch alle $\mathcal{F}_\ell[k]$ mit i< k < j.

  $\mathcal{F}_4[1]$ und $\mathcal{F}_\ell[3]$ stimmen in den ersten 3 Ereignissen überein, so auch $\mathcal{F}_\ell[2]$.



Kandidatengenerierung -- Idee

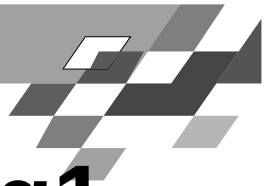
- Aus häufigen Episoden sollen um eins längere Episoden generiert werden.
- Die längste Abfolge von Sequenzen i=1,...,m mit denselben ℓ-1 Ereignissen heißt ein Block.
- Innerhalb eines Blockes werden alle Episoden (an ℓter Stelle) kombiniert, um solche der Länge ℓ+1 zu generieren.

$\mathcal{F}_\ell$

i,j↓ ℓ→	1	2...	ℓ
1	A	B	C
...			
m	A	B	F
m+1	A	C	D

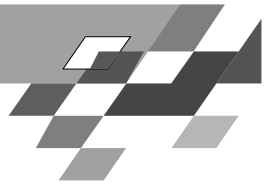
→ $C_{\ell+1}$

$\mathcal{F}_\ell$.blockstart[1]=1
$\mathcal{F}_\ell$.blockstart[2]=1
...
$\mathcal{F}_\ell$.blockstart[m]=1
$\mathcal{F}_\ell$.blockstart[m+1]=m+1

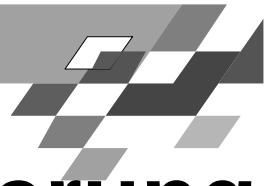


WINEPI: Kandidatengenerierung1

- Gegeben ein sortiertes Array $\mathcal{F}_l$ von häufigen parallelen Episoden der Länge l
- Finde ein sortiertes Array paralleler Episoden der Länge $l+1$ als Kandidaten.

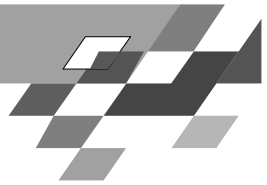


1. $C_{\ell+1}$:={ };
2. k:=0;
3. If ℓ=1 then for x:=1 to $|\mathcal{F}_\ell|$ do $\mathcal{F}_\ell$.blockstart[h]=1;
4. For i:=1 to $|\mathcal{F}_\ell|$ do /*Ein i nach dem anderen durchgehen */
5. Current_blockstart:=k+1;
6. For (j:=i; $\mathcal{F}_\ell$.blockstart[i]= $\mathcal{F}_\ell$.blockstart[j];j:=j+1) do /*j läuft */
7. For x:=1 to ℓ do α[x]:= $\mathcal{F}_\ell$[i][x]; α[ℓ+1]:= $\mathcal{F}_\ell$[j][ℓ];
8. For y:=1 to ℓ-1 do /* Unterepisoden sollen in $\mathcal{F}_\ell$ vorkommen*/
9. For x:=1 to y-1 do β[x]:= α[x];
10. For x:=y to ℓ do β[x]:= α[x+1];
11. If β ist nicht in $\mathcal{F}_\ell$, then gehe zum nächsten j in Zeile 6, else speichere α als Kandidat.
12. k:=k+1;
13. $C_{\ell+1}$[k]:=a;
14. $C_{\ell+1}$.blockstart[k]:=current_blockstart;
15. Output $C_{\ell+1}$;



Komplexität der Kandidatengenerierung

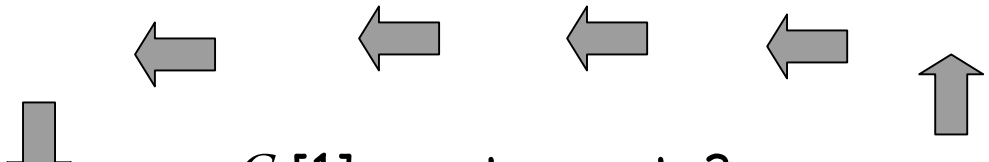
- Theorem: Die Kandidatengenerierung hat die Komplexität $O(\ell^2 |\mathcal{F}_\ell|^2 \log |\mathcal{F}_\ell|)$.
- Beweis: Zeile 3 braucht $O(|\mathcal{F}_\ell|)$.
  Die äußere Schleife (Zeile 4) wird $O(|\mathcal{F}_\ell|)$ mal durchlaufen.
  Die innere Schleife (Zeile 6) wird $O(|\mathcal{F}_\ell|)$ mal durchlaufen.
  In den Schleifen werden Kandidaten (Zeile 7) und Unterepisoden (Zeile 8-10) konstruiert in der Zeit $O(\ell+1+\ell(\ell-1))$.
  Die ℓ-1 Unterepisoden werden in $\mathcal{F}_\ell$ gesucht (Zeile 11). Da $\mathcal{F}_\ell$ sortiert ist, gelingt dies in $O(\ell \log |\mathcal{F}_\ell|)$.
  $O(|\mathcal{F}_\ell| + |\mathcal{F}_\ell| |\mathcal{F}_\ell| (\ell^2 + \ell(\ell-1)) \ell \log |\mathcal{F}_\ell|) = O(\ell^2 |\mathcal{F}_\ell|^2 \log |\mathcal{F}_\ell|)$.
  Q.e.d.



Datenbankdurchlauf -- Idee

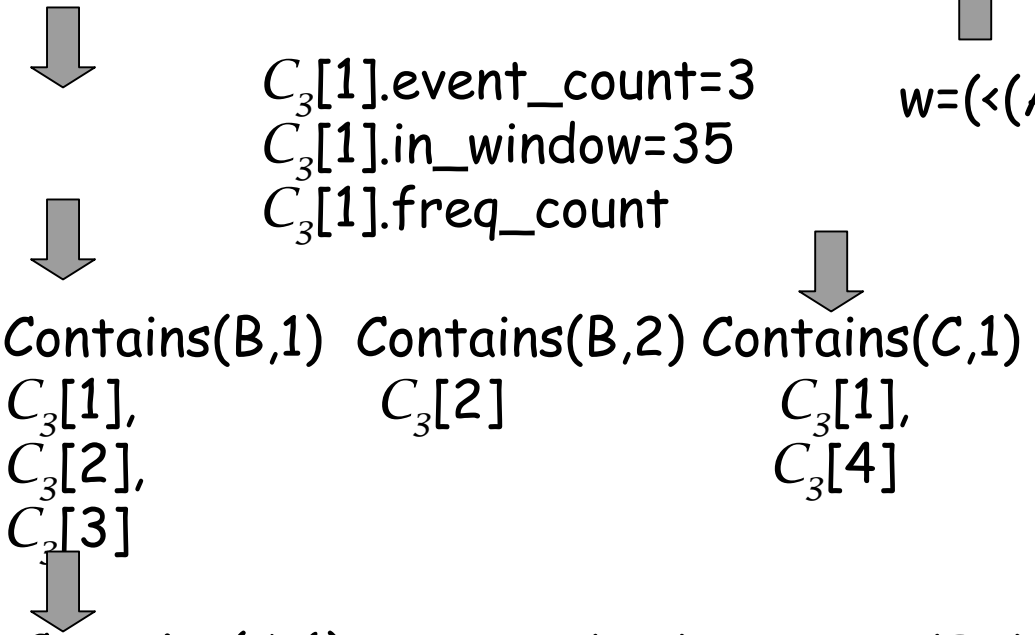
- Contains(A,a) enthält alle Episoden, in denen der Ereignistyp A genau a mal vorkommt. So werden parallele Episoden über ihre Attribute indexiert.
- α.event_count speichert, wie viele Ereignisse von α in Fenster w vorkommen.
- Wenn $|\alpha|$ Ereignisse in w vorkommen, speichern wir ts von w in α.in_window. Das war der Anfang eines Fensters mit der vollständigen Episode.
- Wenn α.event_count abnimmt, wird α.freq_count um die Anzahl von Fenstern erhöht, in denen die gesamte Episode vorkam, d.h. α.event_count = $|\alpha|$. So wird bei jeder Episode hochgezählt, in wie vielen Fenstern sie vorkam.

Beispiel



C_3[1].event_count=3
C_3[1].in_window=35
C_3[1].freq_count

w=(<(A,35), (B, 37), (C,38),(E,39)>, 35,40)



Contains(B,1) C_3[1], C_3[2], C_3[3]

Contains(B,2) C_3[2]

Contains(C,1) C_3[1], C_3[4]

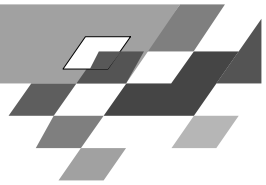
Contains(A,1) C_3[1], C_3[2], C_3[3], C_3[4]

Contains(D,1) C_3[4]

Contains(F,1) C_3[3]

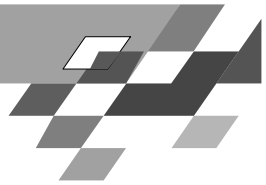
C_3

	1	2	3
1	A	B	C
2	A	B	B
3	A	B	F
4	A	C	D



Update der Fenster

- Beim Verschieben der Fenster von w nach w' bleiben die meisten Ereignisse dieselben: nur ein Ereignis kommt hinzu und ein Ereignis verschwindet.
  - Alle Episoden mit dem neuen Ereignistyp A können über contains(A,1) erreicht und ihr event_count um 1 erhöht werden.
  - War bereits ein Vorkommen von A in Fenster w, so können die passenden Episoden über contains(A,2) erreicht und ihr event_count um 1 erhöht werden.



Datenbankdurchlauf

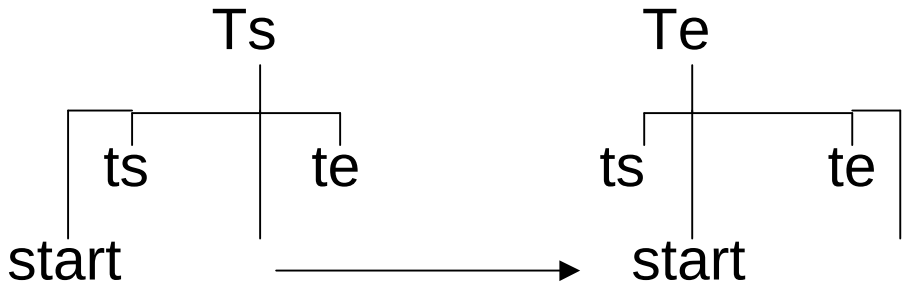
- Gegeben: Eine Sammlung von Episoden C, eine Ereignissequenz **s**=(s, Ts, Te), eine Fensterbreite win, eine Häufigkeitsschranke min_fr.
- Finde die Episoden von C, die häufig in **s** vorkommen bzgl. win und min_fr.

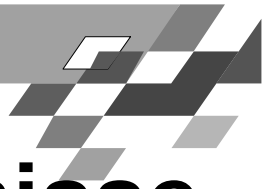
Datenbankdurchlauf1: Initialisierung

```
1.  For each α in C do
2.      For each A in α do                    /* Initialisieren mit 0 */
3.          A.count:=0;
4.          For i:=1 to | α | do contains(A,i):={ };
5.  For each α in C do                        /* Struktur aufbauen */
6.      For each A in α do
7.          a:=Anzahl von Ereignissen des Typs A in α;
8.          contains(A,a):=contains(A,a) ∪ {α};
9.      α.event_count:=0;                     /* Initialisieren mit 0 */
10.     α.freq_count:=0;
```

Datenbankdurchlauf2: neue Ereignisse

1. For start:=Ts – win+1 to Te do /* neue Ereignisse in w' */
2. For all (A, t) in s mit t=start+win – 1 do
3. A.count:=A.count+1;
4. For each α in contains(A,A.count) do
5. α.event_count:= α.event_count+A.count;
6. If α.event_count= | α | then α.in_window:=start;





Datenbankdurchlauf3: alte Ereignisse

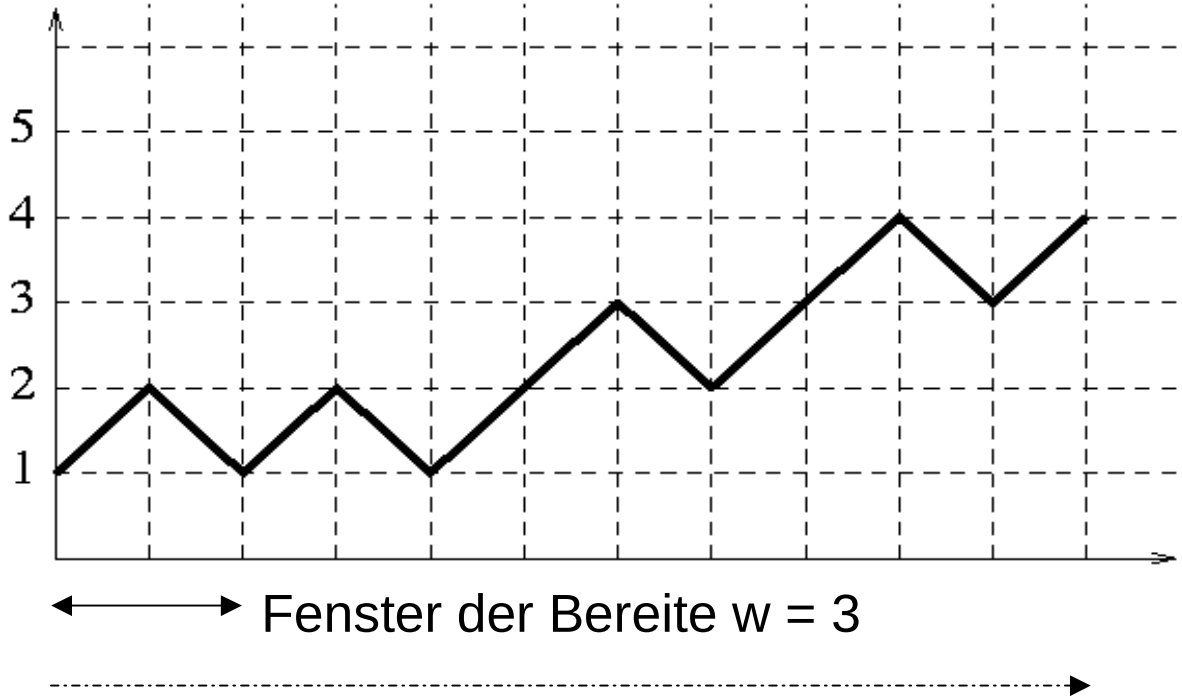
1. For all (A, t) in s mit t=start – 1 do
2. For each α in contains(A,A.count) do
3. If α.event_count= | α | then
4. α.freq_count:= α.freq_count- α.in_window+start;
5. α.event_count:= α.event_count – A.count;
6. A.count:=A.count – 1;
7. For all Episoden α in C do /* Ausgabe*/
8. If α.freq_count/(Te-Ts+win-1) min_fr then output α;

Komplexität des Datenbankdurchlaufs

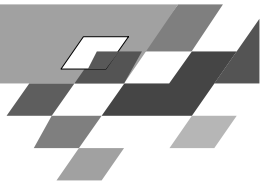
- Theorem: Die Komplexität des Datenbankdurchlaufs für parallele Episoden ist $O((n+\ell^2)\,|C|)$, wobei alle Episoden die Länge ℓ haben und n die Länge der Sequenz ist.
- Initialisierung braucht $O((n+\ell^2)\,|C|)$.
  In den innersten Schleifen bei neuen Ereignissen (Zeile 4) und bei alten Ereignissen (Zeile 5) wird so oft auf α.event_count zugegriffen wie sich das Fenster verschiebt: $O(n)$. Dies kann allen Episoden passieren: $|C|$.
  Der update wegen neuer und alter Ereignisse braucht also $O(n\,|C|)$.
  Q.e.d.

Clustering Vorbereitung

Zeitreihe $s = (x_1,...,x_n)$ in Subsequenzen $s_i = (x_i,...,x_{i+w-1})$ aufteilen



⇨ Schritt 2



Clustering

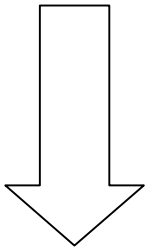
Distanzmaß $d(s_i,s_j)$: Entfernung zwischen zwei Subsequenzen
Bsp.: Euklidischer Abstand $(\Sigma(x_i-y_i)^2)^{0,5}$
Konstante $d > 0$: gibt an, wie groß der Unterschied zwischen den Subsequenzen sein darf

a1= 

a2= 

a3= 

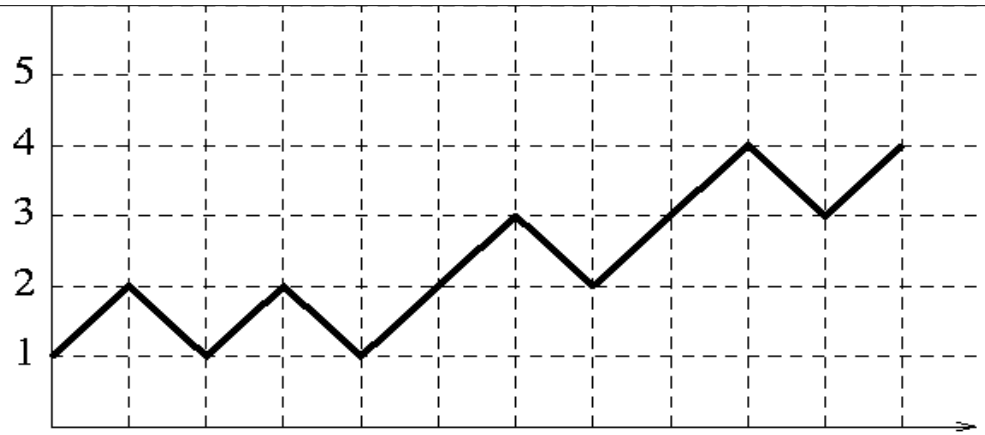
Bilde aus der Menge aller Subseqenzen Cluster $C_1,...C_k$

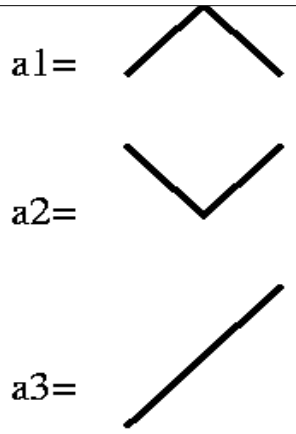


Jedes Cluster erhält ein Symbol $a_1,..a_k$ („Shapes")

Anwendung des Clustering

Die Serie $s = (x_1,\ldots,x_n)$ kann jetzt mit Hilfe der shapes beschrieben werden („diskretisiert")



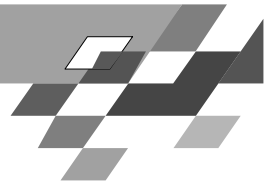


Original time series = (1, 2, 1, 2, 1, 2, 3, 2, 3, 4, 3, 4)

Window width = 3

Discretized series = (a1, a2, a1, a2, a3, a1, a2, a3, a1, a2)

Primitive shapes after clustering



Regeln in diskreten Sequenzen

- Regeln der Form Wenn A auftritt, dann tritt B in der Zeit T auf einfach ableitbar mithhilfe APRIORI
- Berechnung in der Zeit $m*k^2$ möglich
  - (k=Anzahl der Symbole, m = #verschiedene Möglichkeiten für T)
- Erweiterung:
  - Wenn A_1 und A_2 und ... und A_h innerhalb der Zeit V auftritt, dann tritt B in der Zeit T auf
  - Microsoft $\downarrow$ (1), Microsoft $\uparrow$ (2) + Intel $\rightarrow$ (2) $\Rightarrow$ IBM $\rightarrow$ (3)
  - Problem: Anzahl der Regeln steigt stark an

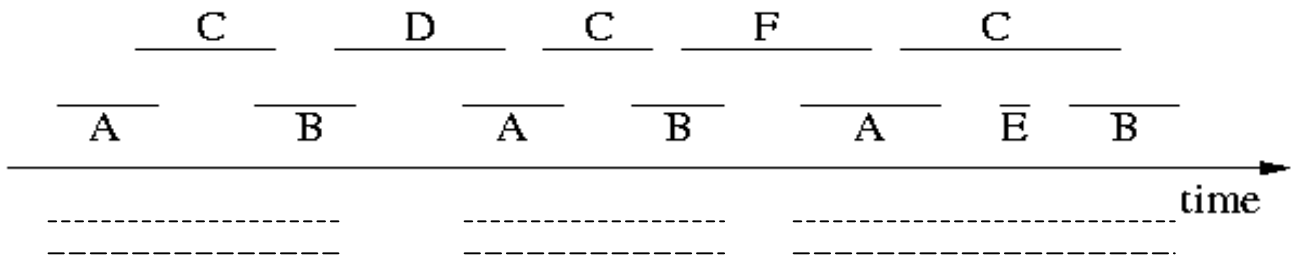
Beziehungen zwischen Ereignissen

- Von James F. Allen wurden 13 verschiedene Intervallbeziehungen festgelegt:
  - A überlappt B, A beendet B, A vor B, A enthält B, ...
- Beispiel: A beendet B

StartB<StartA, EndeA = EndeB,

Beziehungen zwischen Zeit-Intervallen lernen [Höppner]

state interval sequence:



Darstellung der Beziehungen als Matrix:

R1

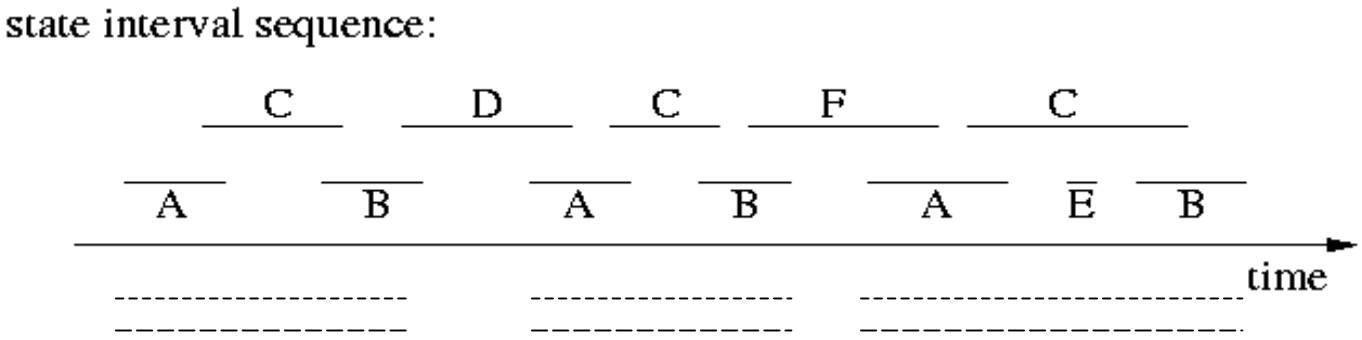
	A	B
A	=	b
B	a	=

R2

	A	B	C
A	=	b	o
B	a	=	io
C	io	o	=

(abbreviations: a=after, b=before, o=overlaps, io=is-overlapped-by)

Regeln



Die Regeln sind von der Form **P** $\rightarrow$ **R**

Prämisse P

	A	B
A	=	b
B	a	=

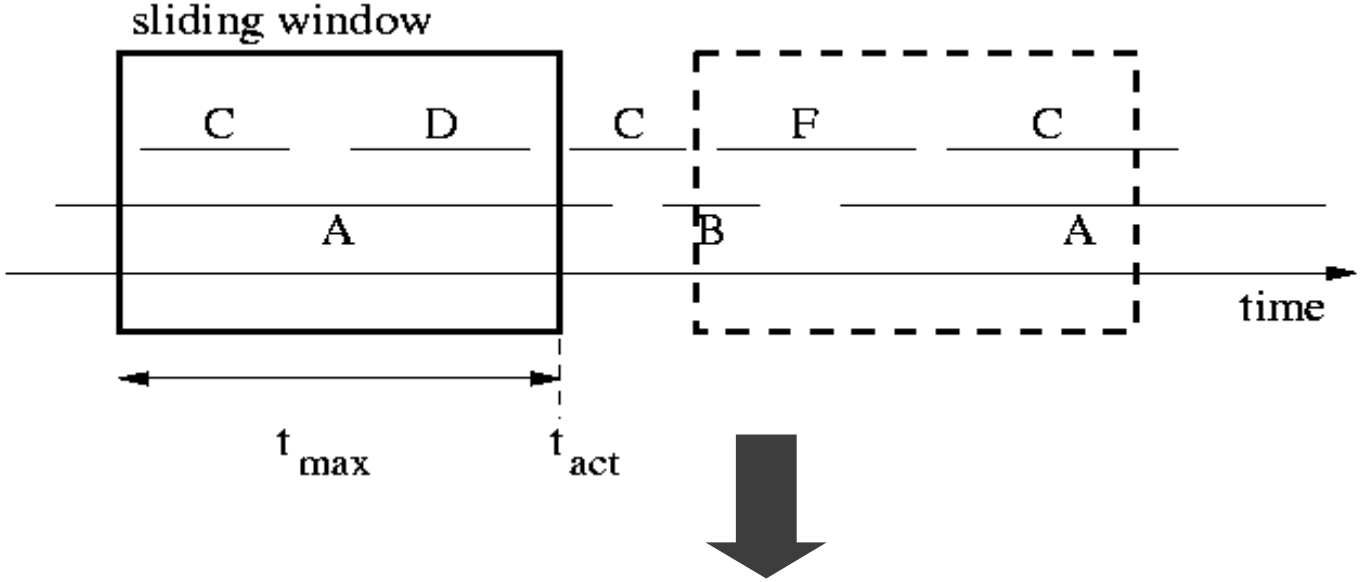
Regel R

	A	B	C
A	=	b	o
B	a	=	io
C	io	o	=

Beispiel: A, B, C sind Verträge verschiedener Kategorien

Häufige Muster finden

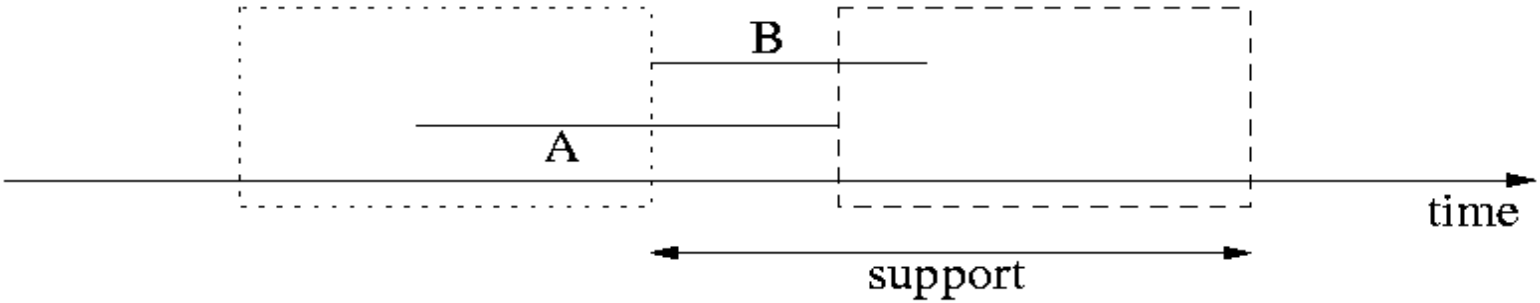
Muster muss im Fenster der Länge t_{max} beobachtbar sein



Der maximale Abstand zwischen den Ereignissen eines Muster ist begrenzt

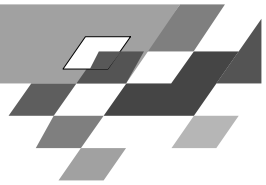
Was bedeutet häufig?

Als Maß für die Häufigkeit von Mustern dient der „Support“



Ein Muster wird als häufig erachtet, wenn es einen Support > $supp_{min}$ hat

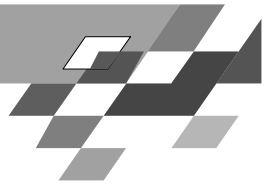
	A	B
A	=	o
B	io	=



Anwendung von APRIORI

- Ermittle den Support aller 1-Muster
- Im k-ten Lauf:
  - entferne alle Muster mit $supp < supp_{min}$
  - generiere aus den verbliebenen k-Mustern eine Menge von Kandidaten für k+1-Muster
  - ermittle den Support der Kandidaten im nächsten Lauf
- Wiederhole diese Schritte, bis keine häufigen Muster mehr gefunden werden können

- Generiere die Regeln aus den häufigen Mustern



Was wissen Sie jetzt?

- Man kann den Apriori Algorithmus für die Entdeckung von Zeitsequenzen anwenden.
- Der Ansatz von Gaudam Das et alii:
  - Fenster werden über die Zeitreihe geschoben
  - Die so erhaltenen Subsequenzen werden durch ein Distanzmaß ge-cluster-t. Es entstehen Muster wie aufsteigend, absteigend.
  - Mit den Mustern als Eingabe werden Assoziationsregeln gelernt.
- Der Ansatz von Frank Höppner:
  - Fenster werden über die Zeitreihe geschoben
  - Matritzen zu Allens Intervallen angelegt
  - Häufige, möglichst lange Sequenzen werden ermittelt und Assoziationsregeln gelernt.