

Data Warehousing

Association Rule Mining

Ulf Leser

Wissensmanagement in der
Bioinformatik



Inhalt dieser Vorlesung

- Association Rule Mining (ARM)
- A-Priori Algorithmus
- Varianten

Aufgabe

- Klassisches Beispiel:
Warenkorbanalyse
- Transaktionen mit binären Attributen
 - Produkt (Item) gekauft oder nicht



- Frage: **Welche Produkte werden häufig zusammen gekauft?**
 - Frequent itemsets
 - Bier und Windeln?
- Data Warehouse Kontext
 - Viele Millionen Transaktionen (T)
 - Viele tausend Produkte (I)
 - Man kann nicht **2^I verschiedene Itemsets** zählen
 - Optimierte Algorithmen notwendig

Genereller

- Definition

*Gegeben eine Datenbank T von Transaktionen, die aus Kombinationen von Items aus der Menge I bestehen. Sei $i, i_1, \dots, i_n \in I$. Eine **Assoziationsregel** ist eine Regel der Form*

$$\{i_1, \dots, i_n\} \rightarrow i$$

- Zweck der Regel: Wer i_1, \dots, i_n kauft, kauft meist auch i
- Beobachtung
 - Wenn $\{i_1, \dots, i_n\} \rightarrow i$ oft auftritt, muss auch $\{i_1, \dots, i_n, i\}$ häufig auftreten
 - Also: Wenn $\{i_1, \dots, i_m\} \subseteq I$ ein häufiges Itemset ist, dann ist jede Regel $\{i_1, \dots, i_{j-1}, i_{j+1}, \dots, i_m\} \rightarrow i_j$ eine **potentielle Assoziationsregel**
- Wenn wir alle häufigen Itemsets finden können, kriegt man die Assoziationsregeln billig dazu
 - Note: Wenn $H = \{i_1, \dots, i_n\} \rightarrow i$, dann ist $H' \rightarrow i$ mit $H' \subseteq H$ **wahr, aber uninteressant**

Support und Confidence

- Regeln gelten meist nicht immer
- ARM sucht Regeln, die **häufig genug** gelten
- Definition

Sei $R=H \rightarrow i$ eine Assoziationsregel und T eine Transaktionsdatenbank:

- Der **Support** von R , $sup(R)$, ist die Häufigkeit von Tupeln in T in denen die Items $H \cup i$ enthalten sind
- Die **Confidence** von R , $conf(R)$, ist definiert als

$$conf(R) = \frac{sup(H \cup i)}{sup(H)}$$

- Bemerkungen
 - Niedriger Support: Spezialregel, geringe Aussagekraft
 - Niedrige Confidence: „Falsche“ Regel, gilt oft nicht
 - „Häufig genug“: Häufiger als ein **Schwellwert minsup**

Beispiel

Transaction	Items
t_1	Bread, Jelly, Peanut Butter
t_2	Bread, Peanut Butter
t_3	Bread, Milk, Peanut Butter
t_4	Beer, Bread
t_5	Beer, Milk

$X \Rightarrow Y$	s	α
Bread \Rightarrow Peanut Butter	60%	75%
Peanut Butter \Rightarrow Bread	60%	100%
Beer \Rightarrow Bread	20%	50%
Peanut Butter \Rightarrow Jelly	20%	33.3%
Jelly \Rightarrow Peanut Butter	20%	100%
Jelly \Rightarrow Milk	0%	0%

Quelle: Dunhum,
Data Mining

Anwendungen

- Überall dort, wo Dinge zusammen passieren und man **Zusammenhänge** vermutet
 - Welche Aktienkurse steigen häufig gemeinsam?
 - Welche Gene werden häufig gemeinsam exprimiert?
 - Welche Ereignisse treten häufig vor einem Systemcrash auf?
 - Welche Wörter treten häufig gemeinsam in interessanten Dokumenten auf?
 - ...

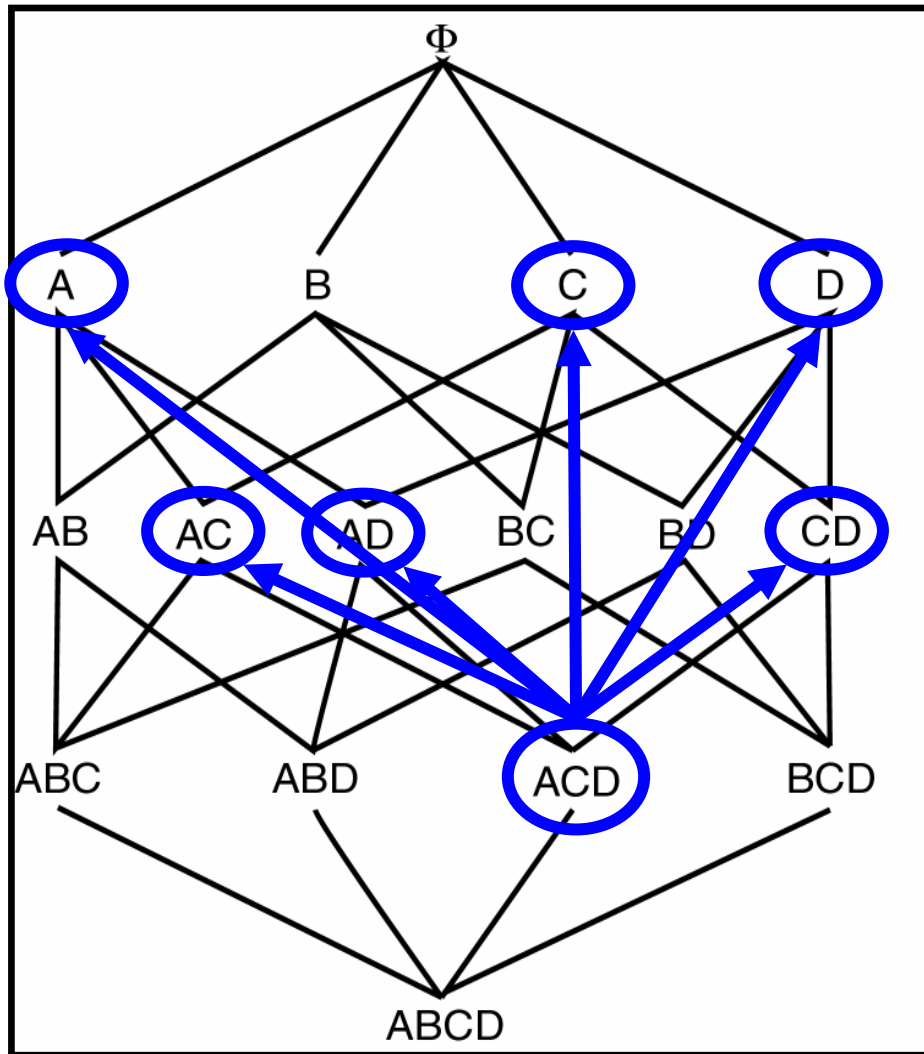
Inhalt dieser Vorlesung

- Einführung Association Rule Mining (ARM)
- A-Priori Algorithmus
 - Grundalgorithmus
 - Kandidatenerzeugung und –pruning
 - Datenstruktur für den Pruningschritt
 - Erweiterung: Partitionierung
 - Ableitung von Assoziationsregeln
- Varianten

Grundidee [AS94]

- Wir wollen nicht 2^l potentiell häufige Itemsets aufzählen und in T zählen
 - Müssen wir auch nicht
- A-Prior-Eigenschaft
 - Downward: Jede Teilmenge eines häufigen Itemsets muss häufig sein
 - Upward: Häufige große Itemsets müssen (nur) aus häufigen kleinen Itemsets bestehen
- Beide Eigenschaften können wir zum zielgerichteten **Aufzählen und Prunen** verwenden
 - Starte mit kleinen Itemsets
 - Upward: Nur die Obermengen häufiger kleiner Itemsets sind potentiell häufige große Itemsets
 - Downward: Von all diesen Obermengen können wir die streichen, die eine nicht-häufige kleinere Teilmenge enthalten

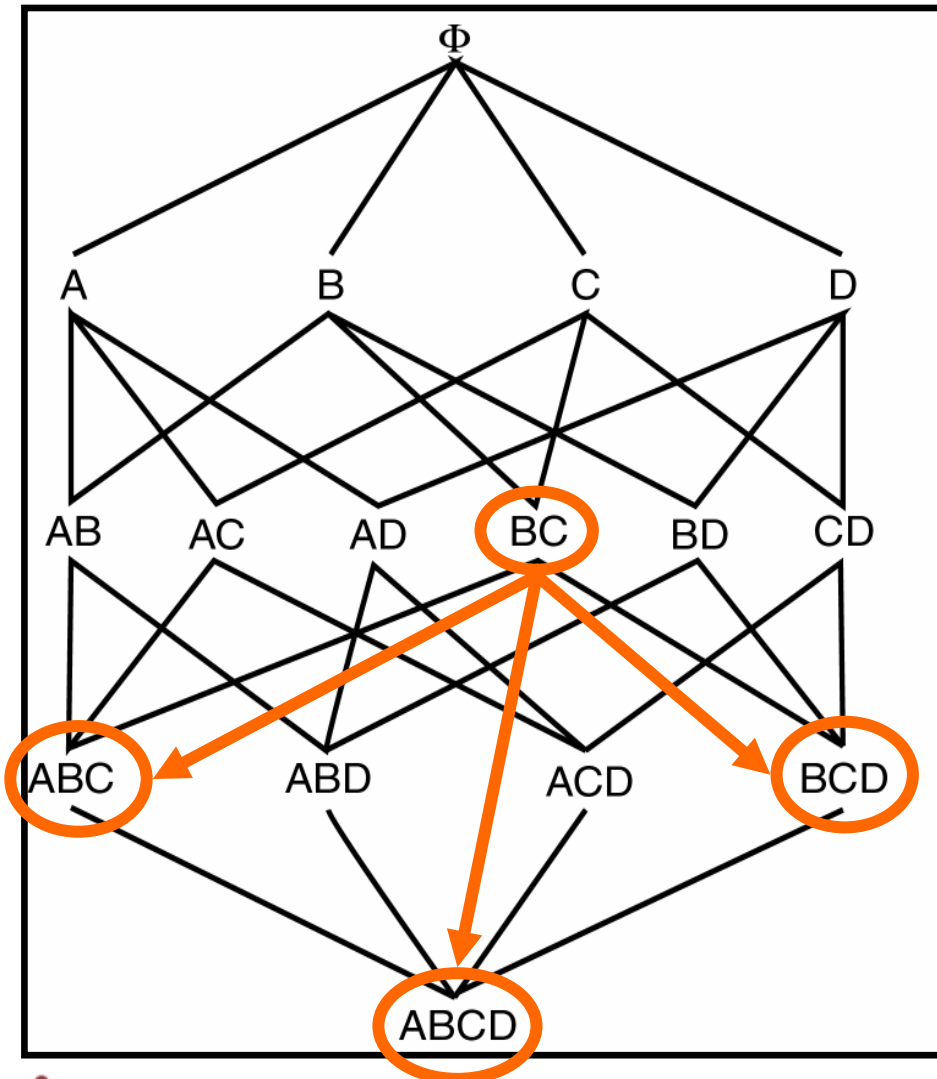
Beispiel



- Downward

- Wenn ACD häufig ist, muss AC, AD, CD, A, C, D häufig sein
- ABCD kann häufig sein

Beispiel



- Downward
 - Wenn ACD häufig ist, muss AC, AD, CD, A, C, D häufig sein
 - ABCD kann häufig sein
- Upward
 - Wenn BC nicht häufig ist, kann ABC, BCD und ABCD nicht häufig sein
 - B und C können häufig sein

Algorithmus: Häufige Itemsets

```
L1 := frequent 1-Itemsets aus I;           // Ein Scan von T
k := 2;                                     // k: Größe der Itemsets
while Lk-1 ≠ ∅ {                             // Noch Obermg sinnvoll?
    Ck := generateCandidates (Lk-1); // später
    for each t ∈ T                             // Viele weitere Scans
        C := frequent(Ck, t);           // Welche Kandidaten
        for each c ∈ C {                       // ... sind häufig?
            c.count++;
        }
    }
    Lk := {c ∈ Ck | (c.count ≥ minsup)}; // Merken
    k++;
}
return ∪k Lk;                             // Alle Häufigen
```

- Benötigt **viele Scans** der Datenbank
- Verbesserungen notwendig (und viele bekannt)

Kandidatengenerierung

- Gegeben Menge L aller häufigen Itemsets mit weniger als k Elementen
- Gesucht: Alle Itemsets mit k Elementen, die **nur häufige Teilmengen** enthalten
 - Zerlegung in zwei Schritte
 - **Join**: Alle Itemsets, die man aus zwei häufigen Itemsets X, Y zusammensetzen kann mit $|X \cap Y| = k-2$ und $x_1 = y_1, \dots, x_{k-2} = y_{k-2}$
 - X und Y stimmen in den ersten k-2 Items überein und unterscheiden sich im (k-1)'ten
 - Vorteil: Benötigt nur Itemsets der Länge k-1, effizient implementierbar, keine falsch Negativen
 - Nachteil: Falsch Positive
 - **Prune**: Für all diese Itemsets teste, ob alle Teilmengen häufig (also in L) sind

Beispiel (minsup=2)

Items
A, C, D
B, C, E
A, B, C, E
B, E

scan

C_1

Itemset	sup
{A}	2
{B}	3
{C}	3
{D}	1
{E}	3

L_1

Itemset	sup
{A}	2
{B}	3
{C}	3
{E}	3

L_2

Itemset	sup
{A, C}	2
{B, C}	2
{B, E}	3
{C, E}	2

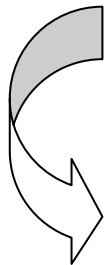
C_2

Itemset	sup
{A, B}	1
{A, C}	2
{A, E}	1
{B, C}	2
{B, E}	3
{C, E}	2

C_2

scan

Itemset
{A, B}
{A, C}
{A, E}
{B, C}
{B, E}
{C, E}



Enumerated	Pruned
{A, C, E}	Nicht aufgezt.
{A, B, C}	Nicht aufgezt.
{B, C, E}	

C_3

Itemset
{B, C, E}

scan

L_3

Itemset	sup
{B, C, E}	2

Quelle: Han/Kamber,
Data Mining

Details des Join

- **Join**: X und Y stimmen in den ersten k-2 Items überein und unterscheiden sich im (k-1)'ten
- Äquivalent zu folgender, einsichtigeren Formulierung
 - Erweitere jeden häufigen k-1 Itemset X mit allen häufigen 1-elementigen Itemsets
 - Das sind Kandidaten der Art $\{x_1, \dots, x_{k-1}, i\}$
 - Entferne die Kandidaten, in denen die Untermenge $\{x_1, \dots, x_{k-2}, i\}$ nicht häufig ist
 - Das ist die Joinbedingung

Inhalt dieser Vorlesung

- Einführung Association Rule Mining (ARM)
- A-Priori Algorithmus
 - Grundalgorithmus
 - Kandidatenerzeugung und –pruning
 - **Datenstruktur für den Pruningschritt**
 - Erweiterung: Partionierung
 - Ableitung von Assoziationsregeln
- Varianten

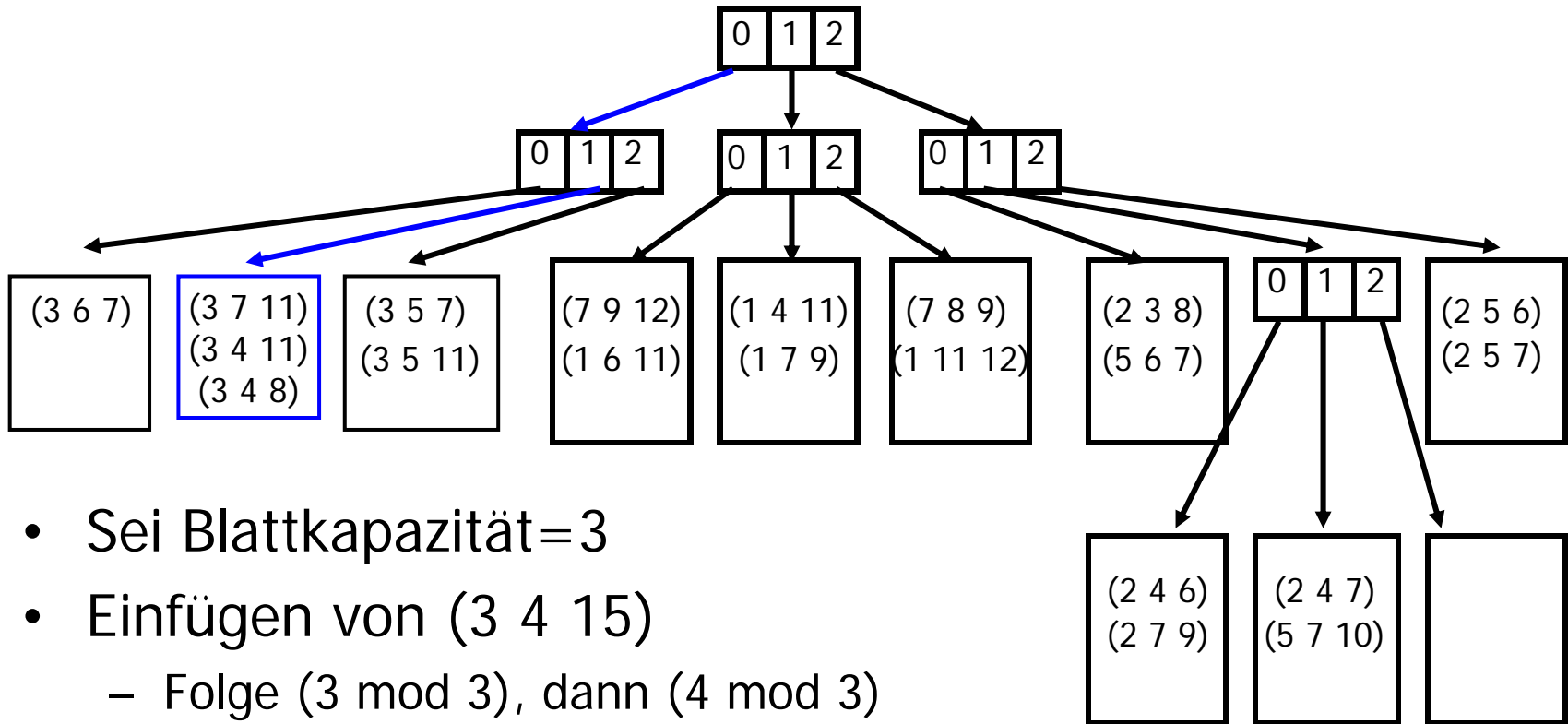
Pruningschritt

- Das **Pruning** ist besonders teuer
 - Gegeben L_{k-1} und Kandidat I_k
 - Wir müssen alle $(k-1)$ -elementigen Teilmengen von I_k aufzählen und in L_{k-1} nachschlagen
 - Nachschlagen muss sehr schnell gehen
- Hashbasierte Mengenindexstruktur
 - Unterstützt die Operation **subSet**(L_{k-1} , I_k) (sei $x=I_k$)
 - Grundidee: Ordne alle Elemente jedes Elements $I \in L_{k-1}$ so in einem Baum an, dass das d -te Element in einem Knoten auf Level d repräsentiert ist
 - Suche x im Baum. An der Wurzel verfolge alle Pfade mit Elementen in x
 - Irgendein Element von x muss das erste Element in einer häufigen Teilmenge sein
 - Auf einem Knoten des Levels d verfolge nur noch Pfade für x_i mit $i > d$
 - Die **d ersten Elemente sind schon im Pfad** bis hierhin gefunden worden
 - Größenbeschränkung: Hashen aller Wert (modulo)
 - Wird eine Teilmenge von x nicht gefunden -> STOP

Hashbaum

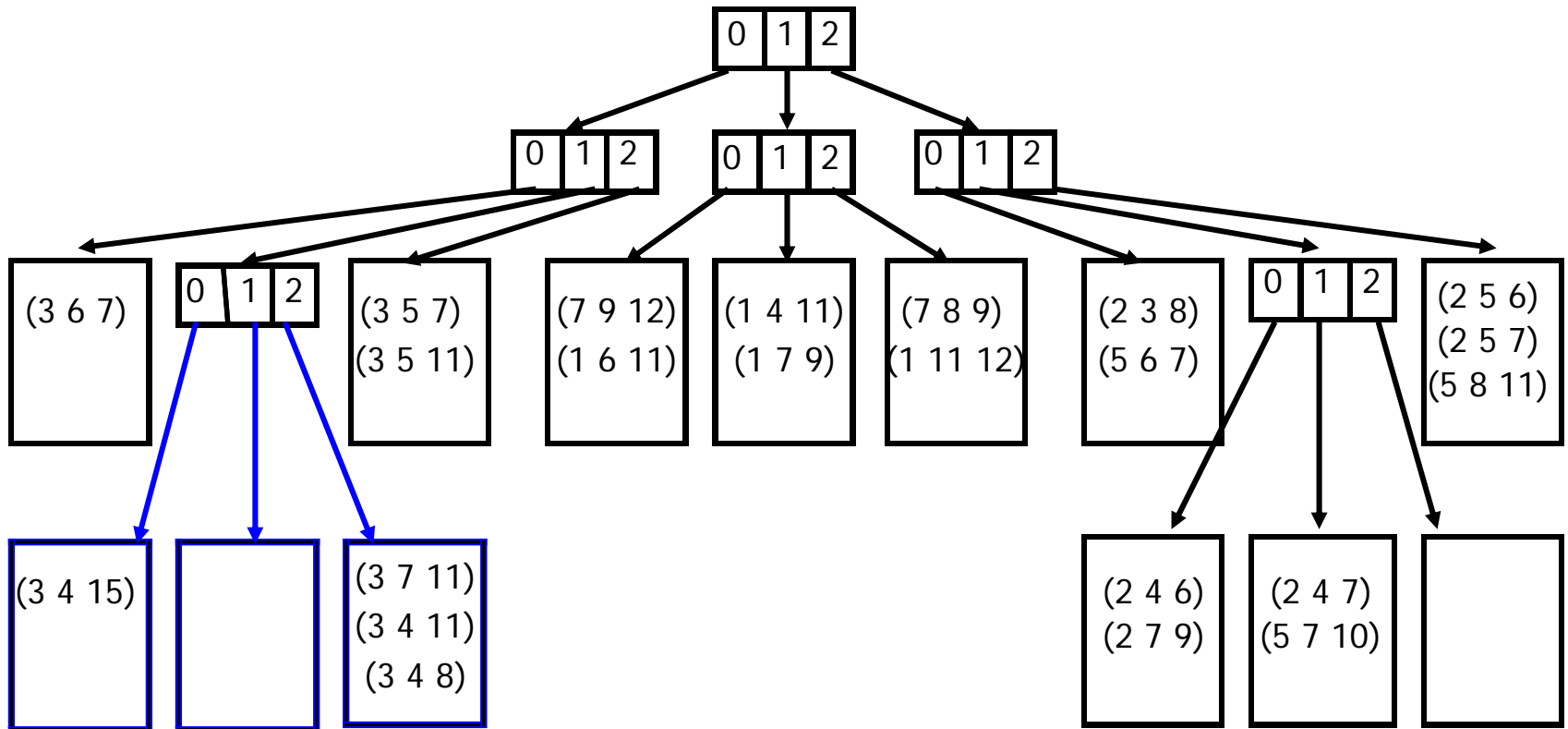
- Datenstruktur: Ein Hasharray, deren Felder baumförmig auf weitere Hasharrays zeigen
- Blätter beinhalten (maximal) k häufige Itemsets
- Innere Knoten haben z Pointer
- Suchen eines Itemset $i = \{i_1, \dots, i_n\}$
 - Starte an der Wurzel, berechne $h(i_1)$, folge dem Pointer
 - An Knoten auf Level d , berechne $h(i_d)$ und folge dem Pointer
 - Wenn Blatt erreicht, alle enthaltenen Itemsets mit i vergleichen
- Einfügen eines Itemsets i
 - Suche i im Baum; nehmen wir an, wir landen in Blatt auf Ebene d
 - Bei Überlauf, Blatt splitten und neuen inneren Knoten für Ebene d erzeugen; sonst i in das Blatt einfügen

Beispiel (mod 3)



- Sei Blattkapazität=3
- Einfügen von (3 4 15)
 - Folge (3 mod 3), dann (4 mod 3)
- Überlauf im Blatt (d=2)
- Splitte gemäß dritter Position

Beispiel (mod 3)

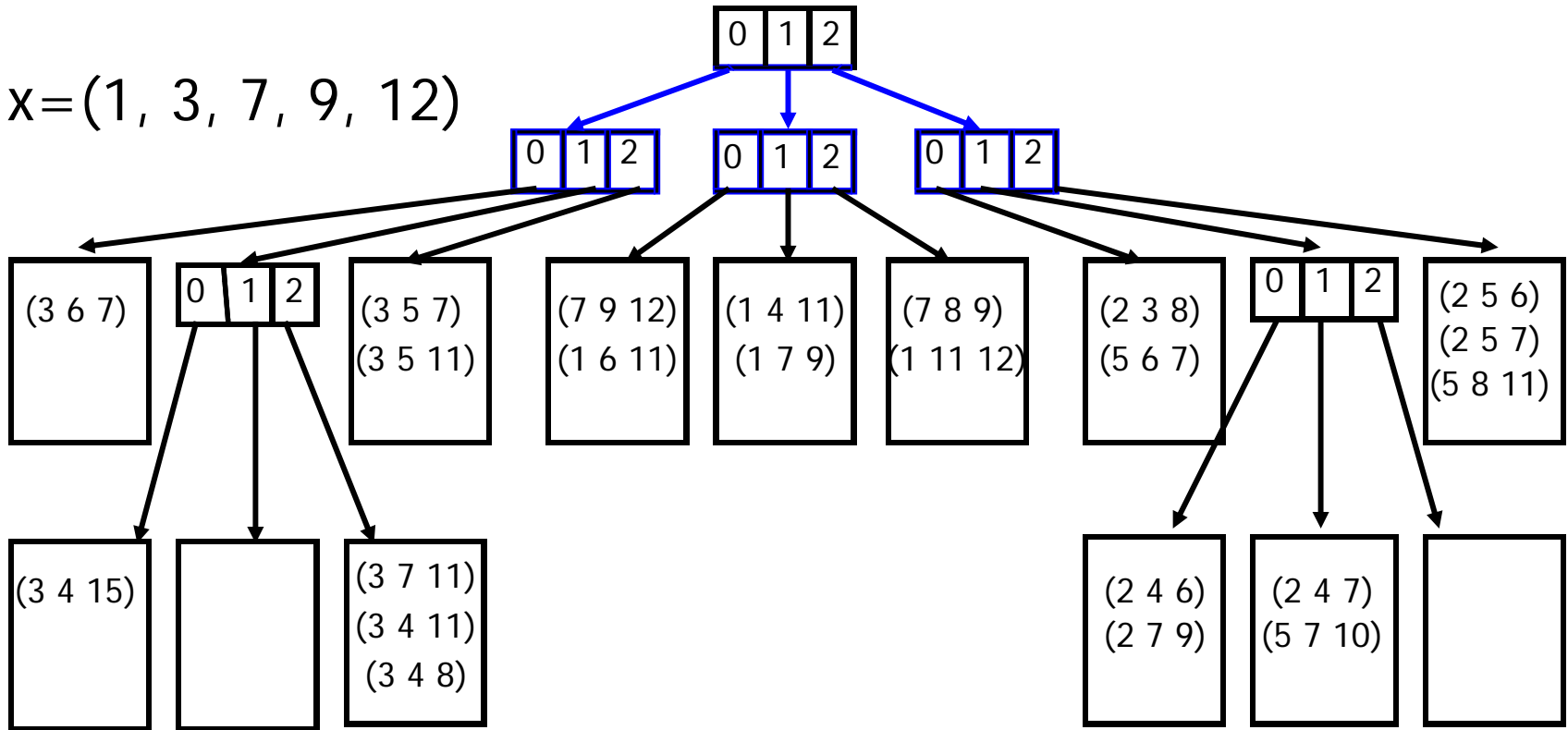


Subset Operation

- Wir wollen testen, ob irgendeine Teilmenge von $x = \{x_1, \dots, x_n\}$ nicht im Baum enthalten ist
- Diese Teilmenge muss **irgendein Element von x als erstes Element** haben
 - Wir wissen nicht welches
 - Alle Werte hashen und alle passenden Kindbäume durchsuchen
- Und irgendein Element als zweites
- Und irgendeines als d'tes
 - An Knoten auf Level d müssen wir dann nur noch mit $x_i, i > d$ weitersuchen
 - Alle vorherigen Knoten sind im Baum auf den früheren Levels „verbraucht“

Beispiel

$x = (1, 3, 7, 9, 12)$

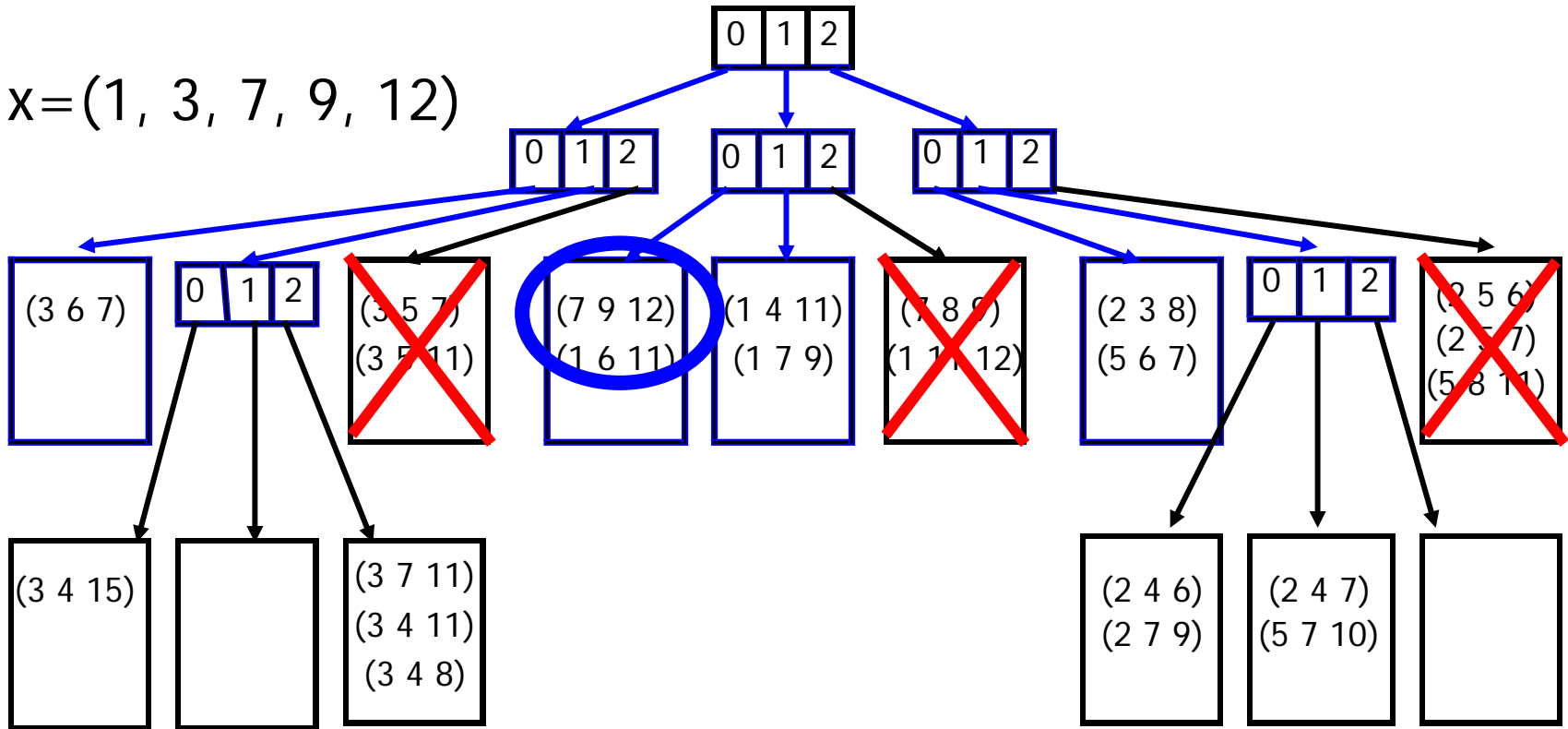


- Wurzelebene 0: $(1 \bmod 3) = 1$, $(3 \bmod 3) = 0$, $(7 \bmod 3) = 2$,

...

Beispiel

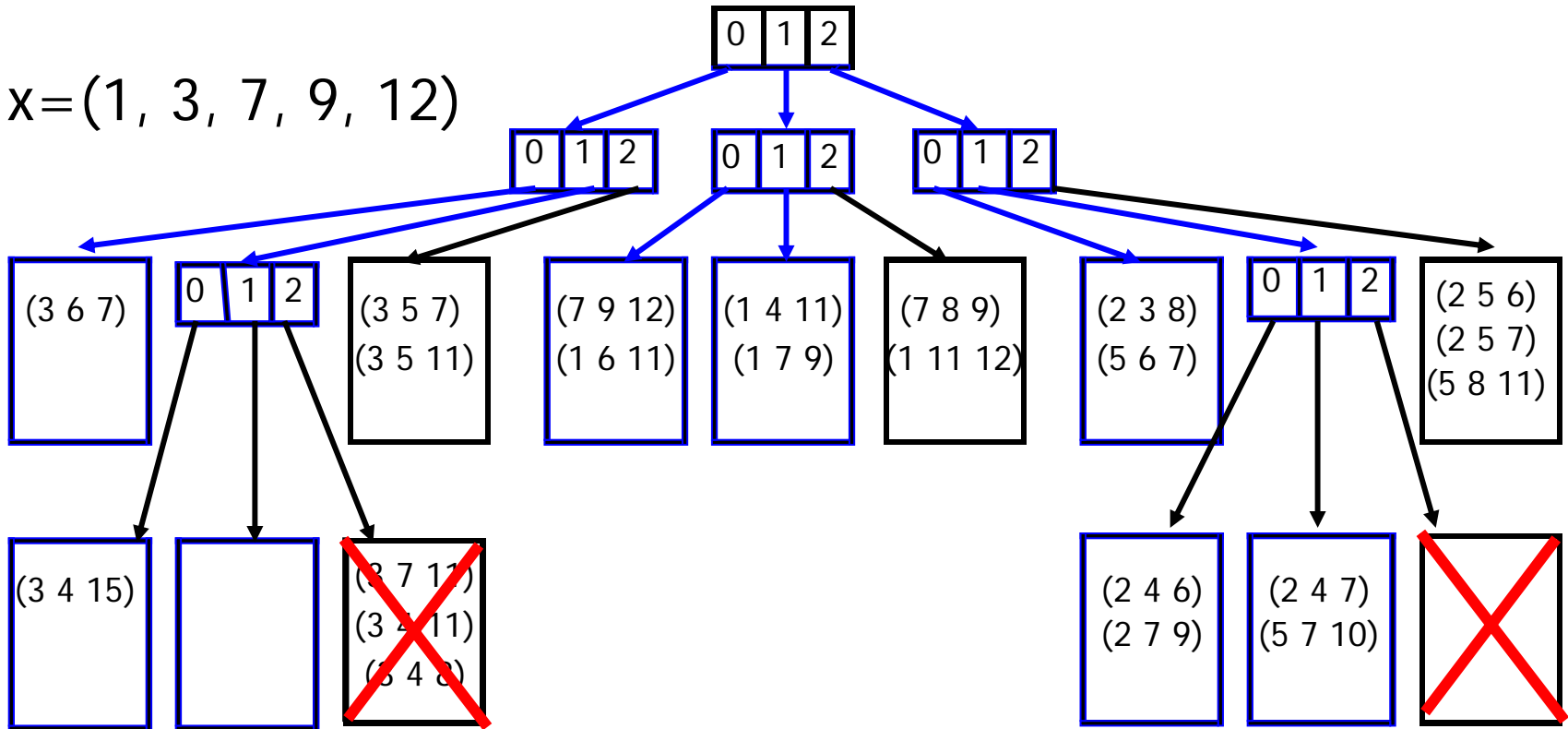
$x = (1, 3, 7, 9, 12)$



- Ebene 1 $(3 \bmod 3) = 0$, $(7 \bmod 3) = 1$, $(9 \bmod 3) = 0$,
 $(12 \bmod 3) = 0$
 - Nur noch Kinder 0 und 1 durchsuchen

Beispiel

$x = (1, 3, 7, 9, 12)$



- Wäre da kein Treffer gewesen ...
- Ebene 2: $(7 \bmod 3) = 1$, $(9 \bmod 3) = 0$, $(12 \bmod 3) = 0$
 - Nur Kinder 0 und 1 durchsuchen

Partitionierung [SON95]

- Vor allem das **häufige Scannen** der Datenbank ist sehr zeitintensiv
- Möglichkeit: Partitioniere das Problem
 - Idee: Zerlege T in k **nicht überlappende Partitionen**
 - Wähle k so, dass jede Partition in den Hauptspeicher passt
 - Phase 1: Zähle alle häufigen Itemsets in jeder Partition
 - Sei der Support relativ zur Datenbankgröße angegeben (x%)
 - Es gilt: Itemset i kann nur häufig in T sein, wenn i **häufig in mindestens einer Partition** ist
 - Suche in einem Scan alle häufigen Itemsets (K) in einer Partition
 - Phase 2: Scanne T und zähle alle Kandidaten aus K
- Vorteile
 - Es werden nur **zwei Scans der Datenbank** benötigt
 - Sehr schnell, wenn Anzahl Kandidaten klein (hoher minsup)
- Nachteil
 - Zweite Phase wird **sehr teuer bei vielen Kandidaten**

Finden der Assoziationsregeln

- Jedes häufige Itemset X kann $|X|$ Regeln „beinhalten“
- Welche davon genügen unseren Ansprüchen?

- Alle Regel der Art $\{x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n\} \rightarrow x_i$ bilden
- Confidence berechnen
 - Da Antezedenz auch ein häufiger Itemset, ist der Support bekannt
- **Kein neues Scannen** der Datenbank notwendig

Inhalt dieser Vorlesung

- Einführung Association Rule Mining (ARM)
- A-Priori Algorithmus
- Varianten
 - Quantitative Regeln
 - ARM mit hierarchischen Itemsets

Motivation

- Bisher haben wir nur binäre Items betrachtet
 - OK für Warenkorbanalyse
 - Nicht OK für Kundenanalyse:
 - „Was haben Kunden gemeinsam, die Versicherungsbetrug verüben?“
 - Keine binären Items
- **Quantitative Assoziationsregeln**

Quantitative Assoziationsregeln

- Attribute haben (numerische) Wertebereiche
- Alter, Einkommen, Zahl der Kinder, Schulden, ...
- Erste Möglichkeit

- Werte vorab binn

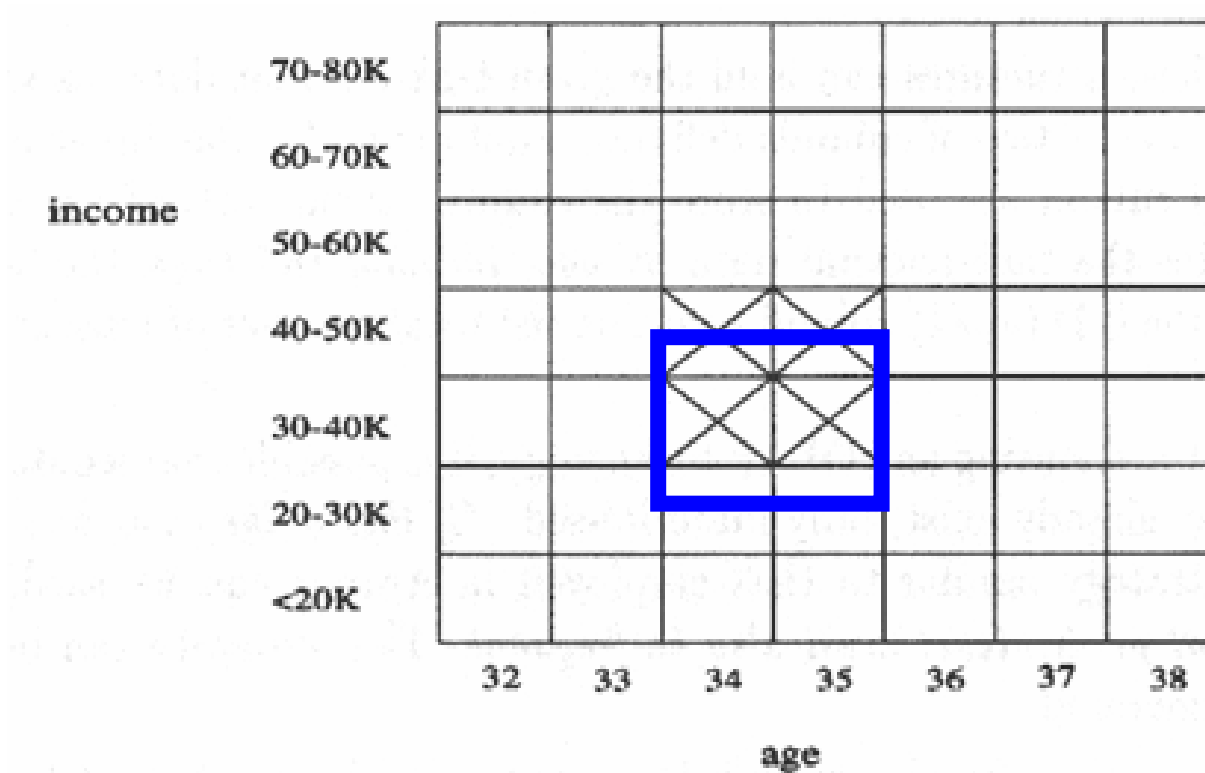
ID	Alter:20..29	Alter:30..39	Alter:40..49
1	1	0	0
2	0	1	0

- Vorteil: Einfach

- Nachteil: Bins bestimmen, was man finden kann

- Viele Bins: **Viele Items**, hohe Auflösung, aber Gefahr, an den Grenzen **interessante Bereiche zu übersehen**
 - Wenig Bins: Weniger Items, grobe Auflösung, geringere (aber immer noch vorhandene) Gefahr, etwas zu übersehen
- Besser: **Dynamische Bestimmung** der Bins aus den Daten während des ARM

Beispiel



- Echter Bereich: $33 < X < 35$, $27500 < Y < 46000$ → Gefahr
- Gefunden: $33 < X < 35$, $30000 < Y < 50000$ → Gefahr

Idee (ohne Details)

- Zunächst **sehr feines Binning** aller Attribute
- Außerdem werden vorab alle zusammenhängenden Bereiche gebildet, die ausreichenden Support in T haben
 - Bins mit zu geringem Support werden nicht entfernt
 - Können zwei benachbarte seltene Bins zu einem häufigen Bereich vereint werden, wird der Bereich zusätzlich aufgenommen
- Alle häufigen Bereiche werden als 1-Itemsets betrachtet und A-Priori gestartet
 - Anpassungen möglich, um die Zusammenhänge zwischen Bereichen eines Attributs auszunutzen
- Nachteil
 - Sehr viele Bereiche, sehr viele Regeln, sehr hohe Laufzeit

Beispiel

ID	Alter	Fam.stand	# Autos
100	23	ledig	1
200	25	verheiratet	1
300	29	ledig	0
400	34	verheiratet	2
500	38	verheiratet	2

Intervall	Integer
20..24	1
25..29	2
30..34	3
35..39	4

ID	Alter	Fam.stand	# Autos
100	1	2	1
200	2	1	1
300	2	2	0
400	3	1	2
500	4	1	2

Itemset	Support
{20..24}	1
{25..29}	2
{30..34}	1
{34..39}	1
{20..29}	3
{30..39}	2
{verheiratet}	3
{ledig}	2
{30..39, verheiratet}	2
...	...

Hierarchische Itemsets

- **Klassifikationsstufen**: Produkte sind unterteilt Klassen, Sparten, Arten, Bereiche, ...
- Nicht nur Regel der Art „SkischuhX1→SkiY1, SkischuhX2 → SkiY1“, sondern auch „Skischuhe→Ski“
 - Oder „Winterbekleidung → Skiausrüstung“
- **Items höher Klassifikationsstufen** haben automatisch wesentlich höheren Support
 - Dafür sind es viel weniger Items
 - Regeln auf unteren Ebenen haben geringeren Support und kommen eventuell nicht über minsup
 - Hoher Support: man findet **abstraktere Regeln**
- **Vorsicht**: Regeln können **redundant** sein
 - Immer, wenn „X→Y“ häufig ist, muss auch „vorfahr(X)→Y“ häufig sein
 - Nur die **spezifischsten**, aber noch häufigen Regeln sind gesucht

Erweiterung des Algorithmus

- Neben den Items werden immer **auch ihre Vorfahren** gezählt
- Tricks
 - Nur die Vorfahren hinzufügen, die in mindestens einem $c \in C_k$ verwendet werden
 - Items, von denen nur noch Oberbegriffe gezählt werden, kann man aus t' löschen

```
L1 := frequent 1-Itemsets aus I;
k := 2;
while Lk-1 ≠ ∅ {
    Ck := generateCandidates (Lk-1);
    for each t ∈ T
        t' = {i | i ∈ t oder i=vorfahr(j ∈ t)};
        C := frequent(Ck, t);           // Welche Kandidaten
        for each c ∈ C {                 // sind häufig?
            c.count++;
        }
    Lk := {c ∈ Ck | (c.count ≥ minsup)}; // Merken
    k++;
}
return ∪k Lk;                         // Alle häufigen
```

Alternative: Top-Down

- Wie bei IceBerg-Cubes
 - Iceberg-Cube: Gruppen mit `having (count (*)) > X`
 - Das ist das selbe Problem wie Itemsets mit `minsup > X`
- Also
 - Zähle Klassifikationsstufen Top-Down auf
 - Nur Items von häufigen Itemklassen müssen näher betrachtet werden
 - „Wenn Schuhe selten sind, müssen auch Winterschuhe selten sein“

Schlusswort: Interessante Regeln

- ARM Algorithmen produzieren oft riesige Regelmengen
 - Gerade hierarchische Itemsets: Die oberen Stufen sind in allen Kombinationen häufig
 - Abhilfe: Stufenweise unterschiedliche minsup
- Viele Regeln sind irreführend
 - 60% der Schüler spielen Fußball, 75% der Schüler essen Schokoriegel
 - 40% der Schüler spielen Fußball und essen Schokoriegel
 - Assoziationsregeln
 - „Spielt Fußball“ → „Isst Schokoriegel“, Konfidenz = 67%
 - TRUE → „Isst Schokoriegel“, Konfidenz = 75%
 - „Fußball spielen“ und „Schokoriegel essen“ sind in Wahrheit negativ korreliert, die Regel hat trotzdem hohe Konfidenz
 - Sehr häufige Items sind in vielen frequent itemsets – das muss aber nichts heißen
- Wie findet man das neue, interessante Wissen?
 - Viele Vorschläge
 - Einfaches Maß:
$$\text{int}(H \rightarrow i) = \frac{\text{sup}(H \cup i)}{\text{sup}(H)} - \text{sup}(i)$$
 - „Was ändert sich für i, wenn H erfüllt ist?“



Literatur

- Zhang, C. and Zhang, S. (2002). "Association Rule Mining - Models and Algorithms", Springer LNAI 2307.
- Agrawal, R. and Srikant, R. (1994). "Fast Algorithms for Mining Association Rules". 20th Int. Conf. Very Large Data Bases.
- Savasere, A., Omiecinski, E. and Navathe, S. (1995). "An Effective Algorithm for Mining Association Rules in Large Databases". 21st Int. Conf. on Very Large Databases, Zurich, Switzerland.