REGUŁY ASOCJACYJNE

Nguyen Hung Son

Wydział Matematyki, Informatyki i Mechaniki
Uniwersytet Warszawski

28.II i 6.III, 2008

Outline

Motywacje

Lista autorów (items)

A	Jane Austen
C	Agatha Christie
D	Sir Arthur Conan Doyle
T	Mark Twain
W	G. Wodehouse

Transakcje

TID	Kupione książki				
10	A	C		T	W
20		C	D		W
30	A	C		T	W
40	A	C	D		W
50	A	C	D	T	W
60		C	D	T	

Znajdź wzorce zachowań klientów., np.

- Co jest często kupopwane? (modne?, sezonowe?)
- Które tytuły są kupione razem?
- Co robić, aby przyciągać klientów?
- ...

W jakiej formie wyrazić wzorzec zachowań klientów?

- macierz kolokacji:

	A	C	D	T	W
A	4	4	2	3	4
C	4	6	4	4	5
D	2	4	4	2	W
T	3	4	2	4	W
W	4	5	3	3	5

- Reguły
  - $A \Longrightarrow C$,
  - $C \Longrightarrow W$,
  - $AC \Longrightarrow T$,
  - $T \Longrightarrow ACW$.

Outline

Podstawowe oznaczenia

- Pozycje (ang. items) opisują dostępne towary. Zakłada się, że zbiorem wszystkich towarów jest $I = \{i_1, i_2, ..., i_m\} \quad (items)$
- baza transakcji $D = \{(tid_1, T_1), (tid_2, T_2)...\}$ zawiera transakcje jako pary (tid_j, T_j), gdzie:
  - tid_j : unikalny identyfikator
  - $T_j \subset I$: zbiór zakupionych towarów .
- itemset: każdy podzbiór zbioru towarów I;
- k-itemset: podzbiór o k elementach.
- $s(X)$ - liczba transakcji zawierających itemset X.

Definition

Regułą asocjacyjną nazywamy każdą implikację typu

$$X \implies Y$$

gdzie X, Y są itemsetami. Jakość takiej reguły mierzymy jest funkcjami:

- wsparcie (support)

$$support(X \implies Y) = s(X \cup Y)$$

- wiarygodność (confidence)

$$confidence(X \implies Y) = \frac{s(X \cup Y)}{s(X)}$$

Przykład

Lista autorów (items)

A	Jane Austen
C	Agatha Christie
D	Sir Arthur Conan Doyle
T	Mark Twain
W	G. Wodehouse

Transakcje

TID	Kupione książki				
10	A	C		T	W
20		C	D		W
30	A	C		T	W
40	A	C	D		W
50	A	C	D	T	W
60		C	D	T	

Reguły	wsparcie	st. wiar.
$A \Longrightarrow C$	4	100%
$C \Longrightarrow W$	5	83,3%
$AC \Longrightarrow T$	4	75%
$T \Longrightarrow ACW$	3	75%

Problem

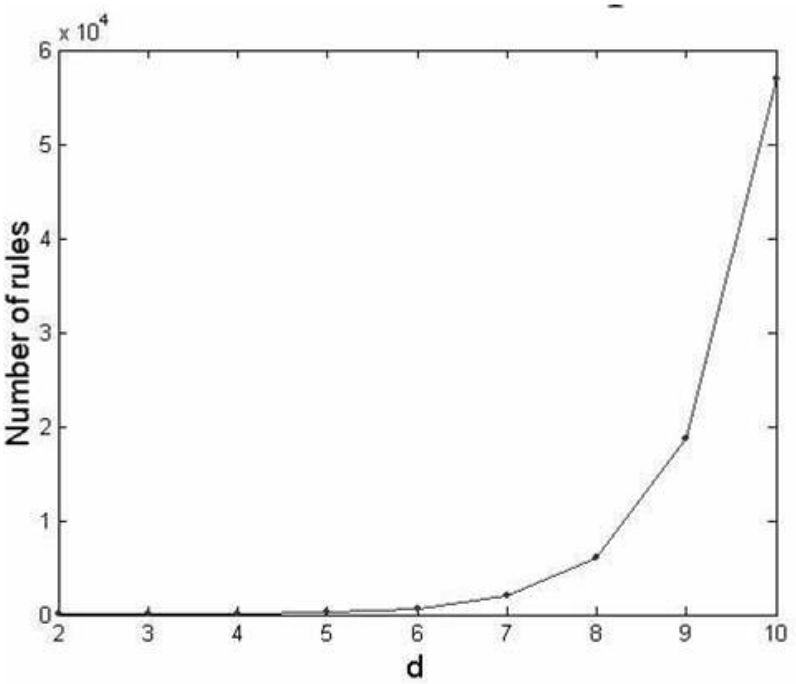
Dane są:

- zbiór pozycji $I = \{i_1, i_2, ..., i_m\}$
- baza transakcji $D = \{(tid_1, T_1), (tid_2, T_2)...\}$
- stałe sup_min = minimalna wartość wsparcia i $conf\_min$ = minimalny stopień wiarygodności

Problem: Znaleźć wszystkie reguły asocjacyjne o
- wsparciu $\geq sup\_min$
- stopniu wiarygodności $\geq conf\_min$

Outline

Problem obliczeniowy



$$R = \sum_{k=1}^{d-1}\left[\binom{d}{k} \times \sum_{j=1}^{d-k}\binom{d-k}{j}\right]$$

$$= 3^d - 2^{d+1} + 1$$

If d=6, R = 602 rules

- Liczba wszystkich reguł asocjacyjnych wynosi $O(3^d)$, gdzie d jest liczbą towarów (items).
- Sprawdzanie wszystkich reguł nie jest wykonywalne!
- Proponowano różne metody szukania z użyciem różnych technik obliczeń: sekwencyjne, równoległe.

Schemat wyszukiwania reguł

Większość istniejących algorytmów działa w dwóch krokach:

Znajdź częste zbiory: znajdź itemsets o wsparciu większym niż min_sup (frequent itemsets).

Podział częstych zbiórów: dla każdego częstego zbioru, znajdź podziały tego zbioru na 2 podzbiory w taki sposób, by powstały reguły o st. wiarygodności większym niż min_conf.

Przykład

Transakcje

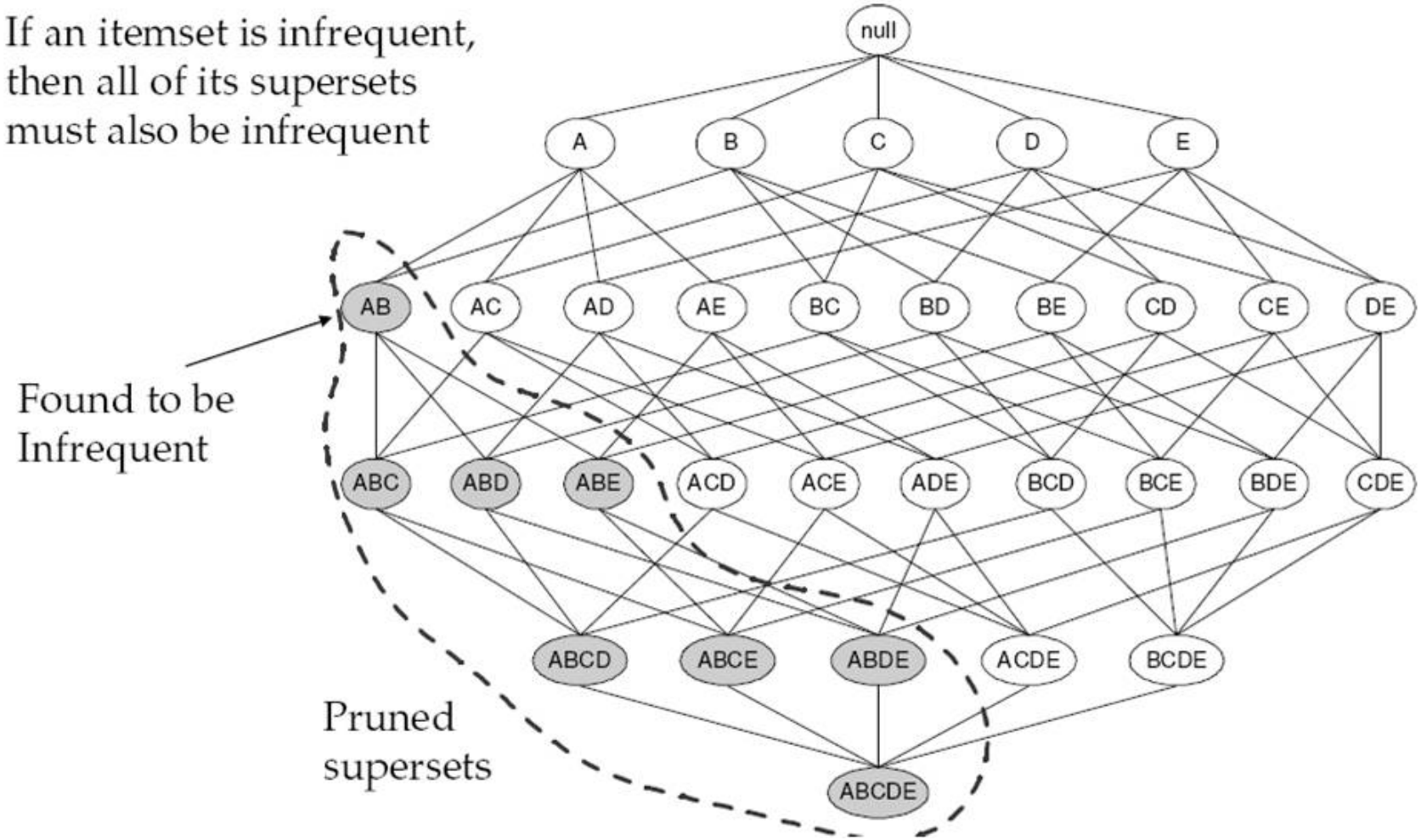
TID	Kupione książki
10	A C T W
20	C D W
30	A C T W
40	A C D W
50	A C D T W
60	C D T

Częste zbiory

wsparcie	Frequent itemsets
100% (6)	C
83% (5)	CW
67% (4)	A, D, T, AC, AW, CD, CT, ACW
50% (3)	AT, DW, TW, ACT, ATW, CDW, CTW, ACTW

- Min. wsparcie = 3 (50%)
- Min. wiarygodność = 75%
- Reguły dla AC:
  $A \Longrightarrow C$ (100% wiarygodności)
  $C \Longrightarrow A$ (66% wiarygodność)

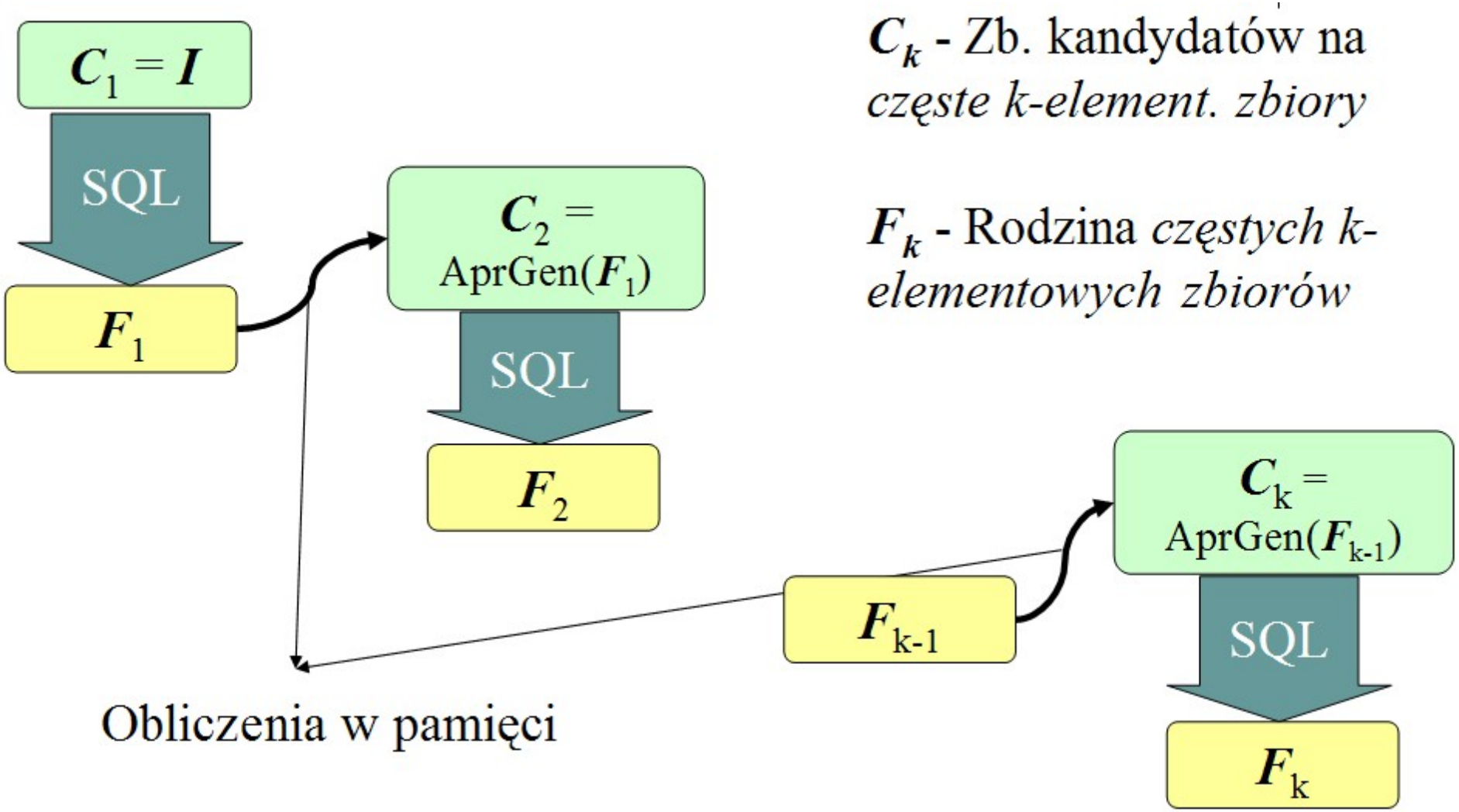
Ważna obserwacja



Wyznaczanie częstych zbiorów

- **Obserwacje:**
  - Jeśli {A,B} jest częstym zbiorem, to {A} i {B} też muszą być częstymi zbiorami.
  - Ogólniej: jeśli X częstym k-elementowym zbiorem, to wszystkie $(k-1)$-elementowe podzbiory X też są częste.
- **Idea:**
  - Znajdź wszystkie 1-elementowe częste zbiory
  - Generuj 2-elementowe częste zbiory z 1-elementowych częstych zbiorów
  - ...
  - Generuj k-elementowe częste zbiory poprzez łączenie $(k-1)$-elementowych częstych zbiorów

Idea algorytmu Apriori



C_k - Zb. kandydatów na *częste k-element. zbiory*

F_k - Rodzina *częstych k-elementowych zbiorów*

Algorytm Apriori

1: $\mathbf{C}_1 := I$; $\mathbf{F}_1 :=$ rodzina 1-elem. zbiorów częstych
2: **for** $(k = 2; \mathbf{F}_{k-1} \neq \emptyset; k++)$ **do**
3: $\quad \mathbf{C}_k :=$ AprioriGen$(\mathbf{F}_{k-1})$;
4: $\quad$ *//generowanie nowych kandydatów*
5: $\quad \mathbf{F}_k := \{X \in \mathbf{C}_k : support(X) \geq min\_sup\}$
6: **end for**
7: Wynik $:= \mathbf{F}_1 \cup \mathbf{F}_2 \cup \mathbf{F}_3 ... \cup \mathbf{F}_k$;

Przykład

- minimalne wsparcie = 3
- ufność = 80%

Transakcje

TID	Kupione książki				
10	A	C		T	W
20		C	D		W
30	A	C		T	W
40	A	C	D		W
50	A	C	D	T	W
60		C	D	T	

Częste zbiory

	Frequent itemsets
$\mathbf{F_1}$	A, C, D, T, W
$\mathbf{C_2}$	AC, AD, AT, AW ...
$\mathbf{F_2}$	AC, AT, AW, CD, CT, CW, DW, TW
$\mathbf{C_3}$	ACT, ACW, ATW, CDW, CTW
$\mathbf{F_3}$	ACT, ACW, ATW, CDW
$\mathbf{C_4}$	

Funkcja AprioriGen($\mathbf{F}_{k-1}$) posiada dwa główne kroki:

- **Łączenie:** do $\mathbf{C}_k$ wstawiamy sumy takich par $X, Y \in \mathbf{F}_{k-1}$, które mają wspólne $k-2$ początkowych elementów. Np. dla

$$\mathbf{F}_{k-1} = \{AB, AC, AD, AE, BC, BD, BE\}$$

mamy

$$\mathbf{C}_k = \{ABC, ABD, ABE, ACD, ACE, ADE, BCD, BCE, BDE\}$$

- **Obcinanie:** Usuwamy z $\mathbf{C}_k$ te zbiory, których nie wszystkie podzbiory $(k-1)$-elementowe są w $\mathbf{F}_{k-1}$.
Np. możemy usuwać ACD, ponieważ CD nie znajduje się w $\mathbf{F}_{k-1}$.
Po obcinaniu otzrymujemy

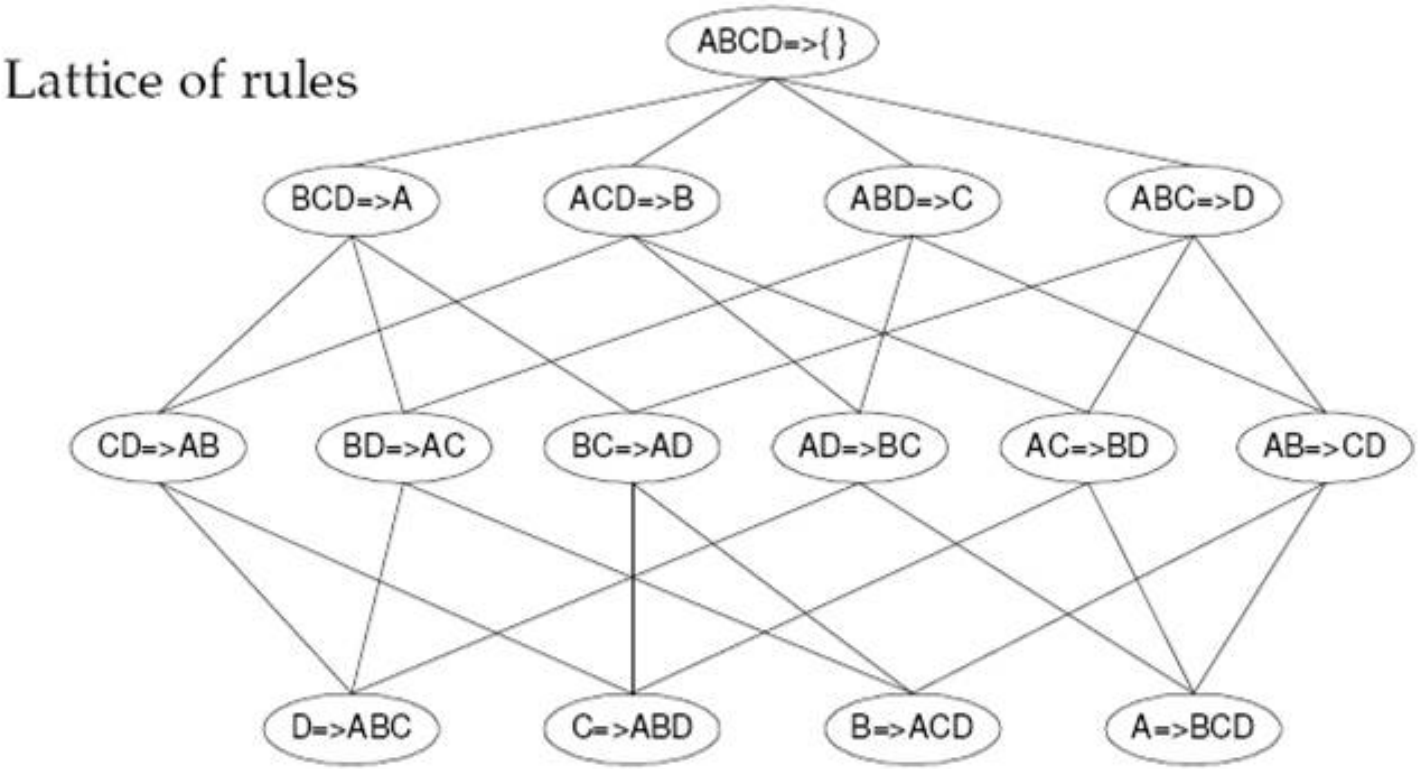
$$\mathbf{C}_k = \{ABC, ABD, ABE\}$$

Generowanie reguł

- **Problem:** Niech X będzie zbiorem częstym. Znaleźć $Y \subset X$ t., że

$$confidence(X \setminus Y \implies Y) > min\_conf$$

- **Obserwacja:**
  "Jeśli AB $\implies$ CD jest wiarygodną regułą, to reguły ABC $\implies$ D i ABD $\implies$ C też są"



Generowanie reguł ze wzorców

Strategie:

- Przerzucać na prawą stronę po kolei pojedyńcze elementy.
- Stosować funkcję $AprioriGen()$ do generowania zbioru warunkowanego Y

Outline

Ulepszenie algorytmu Apriori

- Algorytm Apriori musi przeglądać całą bazę danych w celu obliczenia wsparcia dla kandydatów
- **Ulepszenie:** nowa struktura, która zawiera wyłącznie transakcje, które mogą wspierać aktualnych kandydatów.
  1. *counting_base*: nowa struktura danych, która jest uaktualniana dla każdego kroku k;
  2. Algorytm *AprioriTid*: oblicza wsparcie dla kandydatów skanując wyłącznie strukturę *counting_base*;

AprioriTid

Wejście: zbiór transakcji D, min_sup - minimalne wsparcie
Wyjście: zbiór wszystkich częstych itemsetów F

//$\mathbf{CB}_k$- zbiór counting_base obliczony w k-tym kroku

1: $\mathbf{C}_1 := I$; $\mathbf{F}_1 :=$ rodzina 1-elem. zbiorów częstych
2: **for** $(k = 2; \mathbf{F}_{k-1} \neq \emptyset; k++)$ **do**
3: $\quad \mathbf{C}_k :=$ AprioriGen$(\mathbf{F}_{k-1})$;
4: $\quad$ *//generowanie nowych kandydatów*
5: $\quad \mathbf{CB}_k =$ Counting_base_generate $(\mathbf{C}_k, \mathbf{CB}_{k-1})$;
6: $\quad$ Support_count$(\mathbf{C}_k, \mathbf{CB}_k)$;
7: $\quad \mathbf{F}_k := \{X \in \mathbf{C}_k : support(X) \geq min\_sup\}$
8: **end for**
9: Wynik $:= \mathbf{F}_1 \cup \mathbf{F}_2 \cup \mathbf{F}_3 ... \cup \mathbf{F}_k$;

Generowanie strukturę *Counting_base*

- $\mathbf{CB}_k$ = skojarzy każdą transakcję t z listą kandydatów występujących w t;
- Elementami $\mathbf{CB}_k$ są pary:

$$(t.TID, \{c \in \mathbf{C}_k | c \subset t\}) = (t.TID, S_k(t.TID))$$

- Jeśli jakaś transakcja nie zawiera kandydatów k-elementowych, to zostanie ona usunieta z $\mathbf{CB}_k$ i ze wszystkich następnych zbiorów *counting_base*;
- Można to wyznaczyć metodą iteracyjną:
  - $\mathbf{CB}_1$:= cała baza transakcji
  - $\mathbf{CB}_k := \{(i, S_k(i))\}$, gdzie $S_k(i)$ powstaje z $S_{k-1}(i)$ w następujący sposób:

$$\text{JEŚLI } \{u_1 ... u_{i-2}, a\} \text{ i } \{u_1 ... u_{i-2}, b\} \in \mathbf{F}_{i-1} \cap \mathbf{CB}_{k-1}$$
$$\text{TO } \{u_1, ... u_{i-2}, a, b\} \in S_k(i)$$

Przykład

D = {(1,acd), (2, bce), (3,abce), (4,be)}.

min_sup = 0.5

Krok 1

counting_base ={(1,{a,c,d}), (2,{b,c,e}), (3,{a,b,c,e}), (4,{b, e}) }

F_1 = {a, b, c, e}

C_2 = {ab, ac, ae, bc, be, ce}

Krok 2

counting_base ={(1,{ac}), (2,{bc,be,ce}), (3,{ab,ac,ae,bc,be,ce}), (4,{be}) }

F_2 = {ac, bc, be, ce}

C_3 = {bce}

Krok 3

counting_base = {(2,{bce}),(3,{bce})}

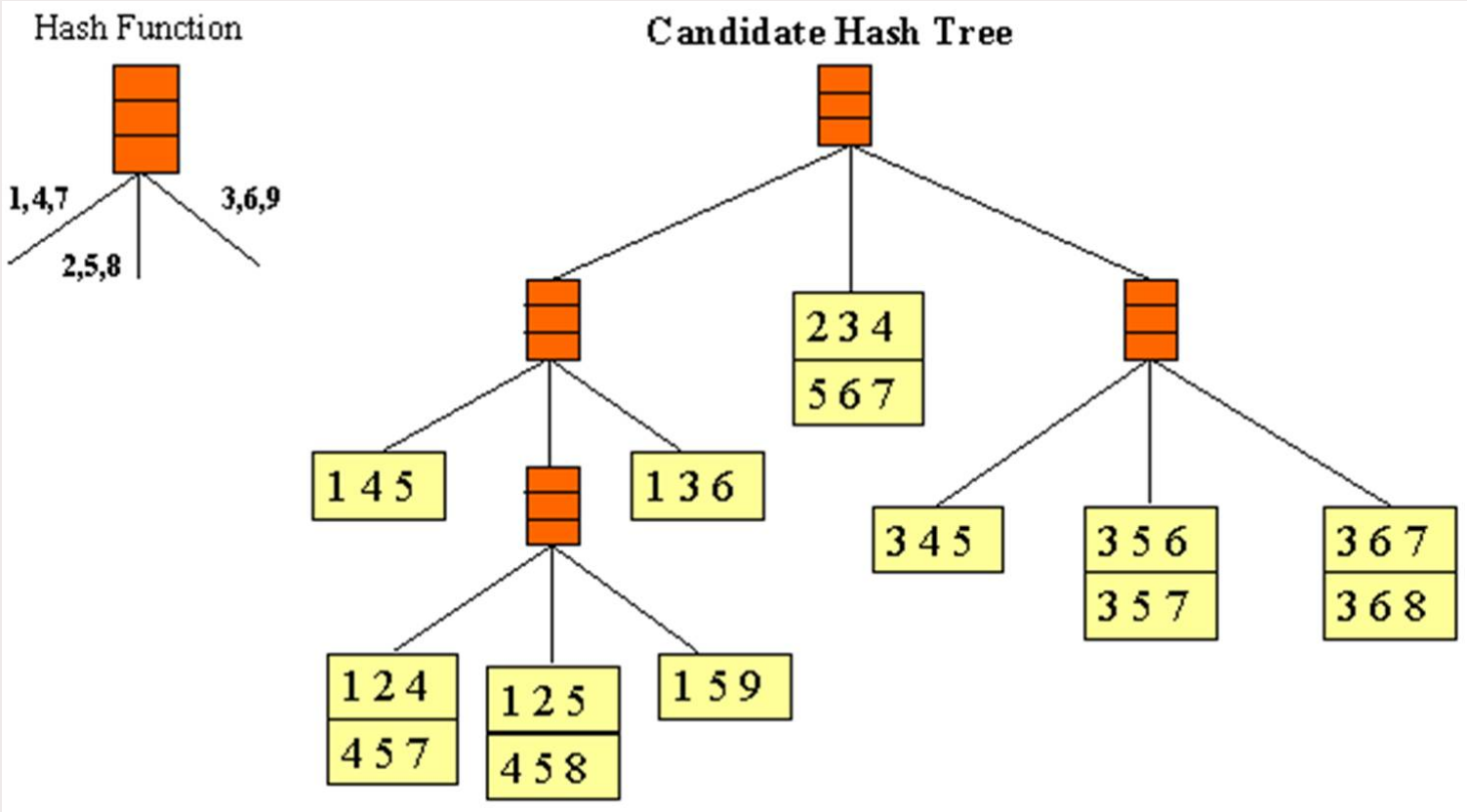
F_3 = {bce}

AprioriHybrid

- AprioriTid przeszukuje tablice $\mathbf{CB}_k$ zamiast skanowaia całą bazę transakcyjną;
  - Jest efektywny wtedy, gdy $\mathbf{CB}_k$ jest dostatecznie mała względem rozmiaru całej bazy.
- AprioriTid jest lepszy od Apriori wtedy, gdy
  - $\mathbf{CB}_k$ mieści się w pamięci;
  - Częste zbiory mają rozkład z "długim ogonkiem"!
- AprioriHybrid
  - wykonuje Apriori w pierwszych iteracjach
  - przełączy na AprioriTid wtedy, gdy spodziewmy, że $\mathbf{CB}_k$ mieści się w pamięci.
- W praktyce, AprioriHybrid może być do 30% szybszy od Apriori i do 60% szybszy niż AprioriTid

Drzewo haszujące dla kandydatów 3-elementowych

1. Utwórz drzewo



2. Dla każdej transakcji, popraw wsparcie dla kandydatów.

Outline

Główne idee

- Baza danych jest zapamiętana w oszczędnej strukturze zwanej FP-tree.
- Częste zbiory są obliczone z tego dzrewa;
- Jest to metoda “Dziel i rządź”;
- Baza danych jest przeskanowana dokładnie 2 razy;
  - piewszy raz: częstość wystąpienia każdego przedmiotu (item);
  - drugi raz: Konstrukcja drzewa FP-tree
- O rząd wielkości szybszy niż Apriori.

Ilustracja

- Baza transakcji ($min\_sup = 3$):

TID	Items
1	f, a, c, d, g, i, m, p
2	a, b, c, f, l, m, o
3	b, f, h, j, o
4	b, c, k, s, p
5	a, f, c, e, l, p, m, n

Ilustracja

- Baza transakcji ($min\_sup = 3$):

TID	Items
1	f, a, c, d, g, i, m, p
2	a, b, c, f, l, m, o
3	b, f, h, j, o
4	b, c, k, s, p
5	a, f, c, e, l, p, m, n

- Po pierwszym skanowaniu całej bazy:

f	c	a	b	m	p	l	o	d	e	g	h	i	j	k	n	s
4	4	3	3	3	3	2	2	1	1	1	1	1	1	1	1	1

- Po usunięciu przedmiotów o wsparciu < 3 mamy posortowaną listę częstych przemiotów (item):

Item	f	c	a	b	m	p
Frequency	4	4	3	3	3	3

- Ponownie skanuje bazę. Dla każdej transakcji:

- Ponownie skanuje bazę. Dla każdej transakcji:
  1. usuwamy nieczęste items,

- Ponownie skanuje bazę. Dla każdej transakcji:
  1. usuwamy nieczęste items,
  2. sortujmy items, i

- Ponownie skanuje bazę. Dla każdej transakcji:
  1. usuwamy nieczęste items,
  2. sortujmy items, i
  3. dodajemy ją do FP-tree

- Ponownie skanuje bazę. Dla każdej transakcji:
  1. usuwamy nieczęste items,
  2. sortujmy items, i
  3. dodajemy ją do FP-tree
- przykład

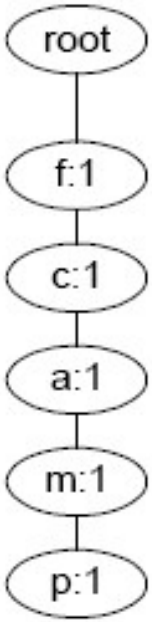
1	f, c, a, m, p
2	f, c, a, b, m
3	f, b
4	c, b, p
5	f, c, a, m, p

- Ponownie skanuje bazę. Dla każdej transakcji:
  1. usuwamy nieczęste items,
  2. sortujmy items, i
  3. dodajemy ją do FP-tree
- przykład

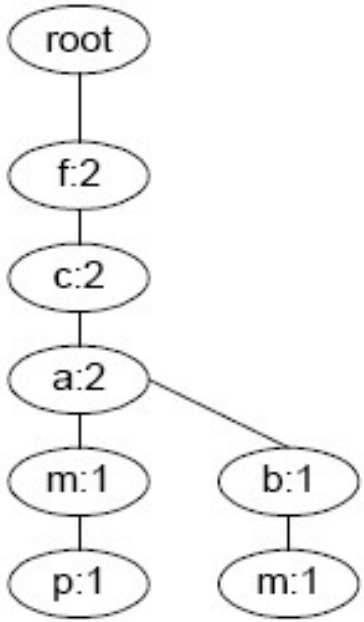
1	f, c, a, m, p
2	f, c, a, b, m
3	f, b
4	c, b, p
5	f, c, a, m, p

- Dodajemy kolejną transakcję do drzewa:

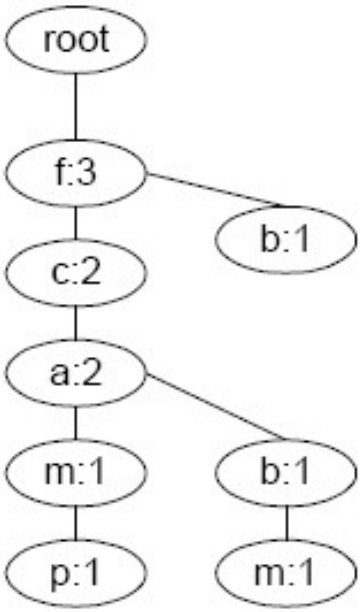
tr1: f, c, a, m, p



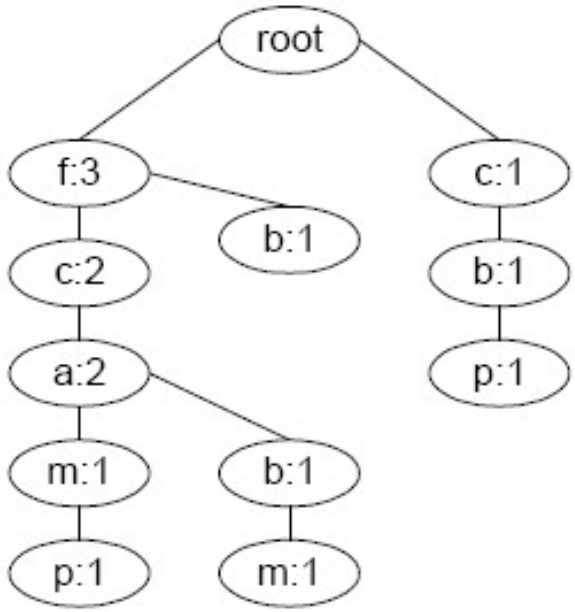
tr2: f, c, a, b, m



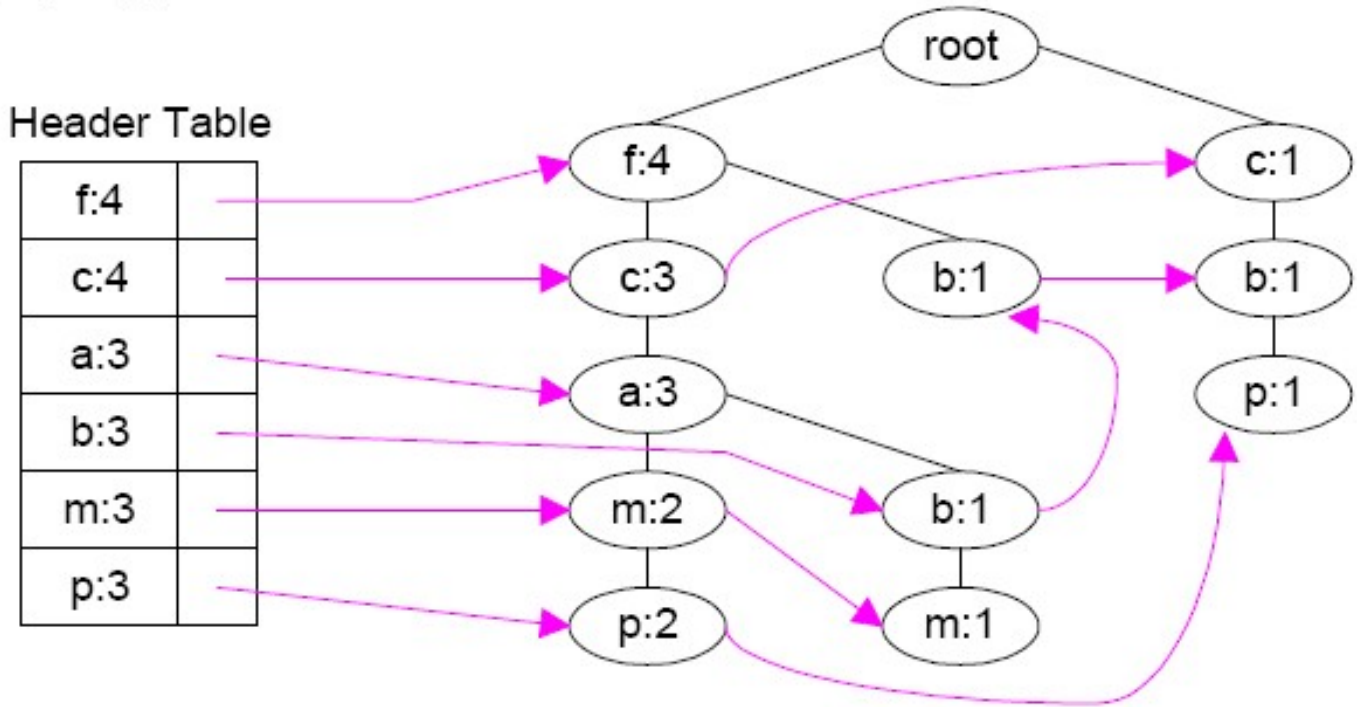
tr3: f, b



tr4: c, b, p



tr5: f, c, a, m, p

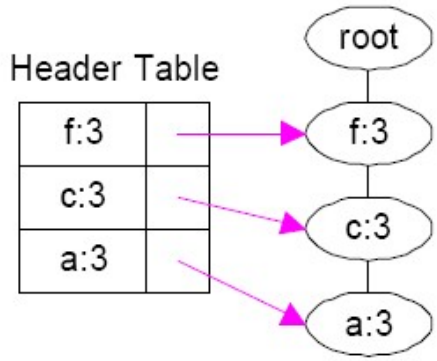


Szukanie częstych zbiorów z drzewa FP-tree

- Proces przeszukiwania sprawdza pojedyńcze przedmioty w tablicy "header table" (od dołu do góry);
- Dla każdego przedmiotu x:
  - Skonstruować bazę warunkowych wzorców;
    - "Ścieżka prefiksowa" (prefix path) = ścieżka prowadząca od korzenia do wierzchołka x;
    - Np. ścieżki prefiksowe dla p: $[f:2, c:2, a:2, m:2]$ i $[c:1, b:1]$;
  - Skonstruować warunkowe drzewo FP-tree
    - Traktujemy bazę warunkowych wzorców dla x jako małą bazę transakcji $D(x)$;
    - Możemy skonstruować FP-tree dla $D(x)$;
    - Jeśli drzewo posiada tylko jedną ścieżkę, to zatrzymujemy i wypisujemy częste zbiory;
    - W przeciwnym przypadku, powtarzamy ten proces.

Przykład

1. Dla p:
   - Baza warunkowych wzorców dla p: [f:2, c:2, a:2, m:2], [c:1, b:1]
   - c jest jedynym częstym przedmiotem i warunkowe FP-tree ma 1 wierzchołek (c:3).
   - Więc {p,c} jest jedynym częstym zbiorem.
2. Dla m:
   - Baza warunkowych wzorców dla p: [f:2, c:2, a:2], [f:1, c:1, a:1, b:1]
   - f, c, a są częstymi przedmiotami



   - warunkowe FP-tree:
   - Odczytujemy częste zbiory: {f,m}, {c,m}, {a,m}, {f,c,m}, {f,a,m}, {c,a,m}, {f,c,a,m}.

Algorytm FP-Growth($Tree$, α)

1: **if** $Tree$ contains a single path P **then**
2: **for** each combination γ of the nodes in P **do**
3: generate pattern $\gamma \cup \alpha$ with support = minimum support of nodes in γ.
4: **end for**
5: **else**
6: **for** each a_i in the header table of $Tree$ **do**
7: generate pattern $\beta = a_i \cup \alpha$ with support = $a_i.support$
8: construct conditional pattern base for β and conditional FP-tree $Tree_\beta$
9: **if** $Tree_\beta \neq \emptyset$ **then**
10: call FP-growth($Tree_\beta, \beta$)
11: **end if**
12: **end for**
13: **end if**