Data Mining

Itemset Mining

Jonas Grab, Christian Ley, Christian Stricker

15. November 2017

Grundidee

Lesen Datensatz ein
Bestimme einelementige Tupel die freq sind
Solange prevFreq nicht leer:
    Baue aus prevFreq alle canidates
    Überprüfe alle canidates, ob alle deren Subtupel freq sind
    Prüfe welche restlichen canidates wirklich freq sind

Optimierung 1

- Codiere jede Zeile vom input binär:
  $1, 0, 1, 1 => 1101_2 = 13_{10}$
  $0, 0, 1, 1 => 1100_2 = 12_{10}$
- Freq-Tupel auch binär codieren:
  Freq Tupel $\{3\}$ in binär: $0100_2 = 4_{10}$
  Freq Tupel $\{1, 4, 5\}$ in binär: $11001_2 = 25_{10}$
- Binäre Operationen wie $'or', 'and', 'equal'$ auf einzelnen Zahlen schneller als bei Arrays/Listen

Optimierung 2

- Bedingung Freq-Tupel: alle seine Sub-Tupel müssen freq sein
- Original: Erzeuge und prüfe alle Sub-Tupel
- Verbesserung: Wie oft wurde next-freq-Tupel aus den prev-freq-Tupel erzeugt:

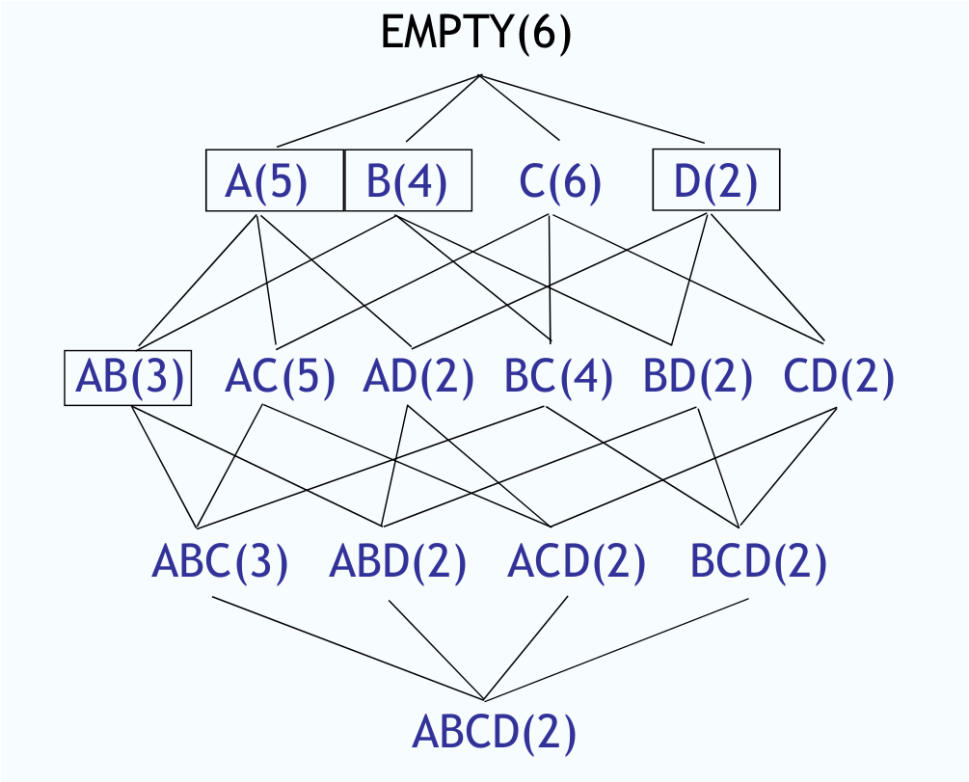
  Bsp.: $\{1,2,3\} = 111_2$ muss 3 mal aus den prev-Tupeln: $011_2 = \{1,2\}, 101_2 = \{1,3\}, 110_2 = \{2,3\}$ erzeugt werden
- Für Iteration k müssen k-mal das next-freq-Tupel erzeugt werden

Border

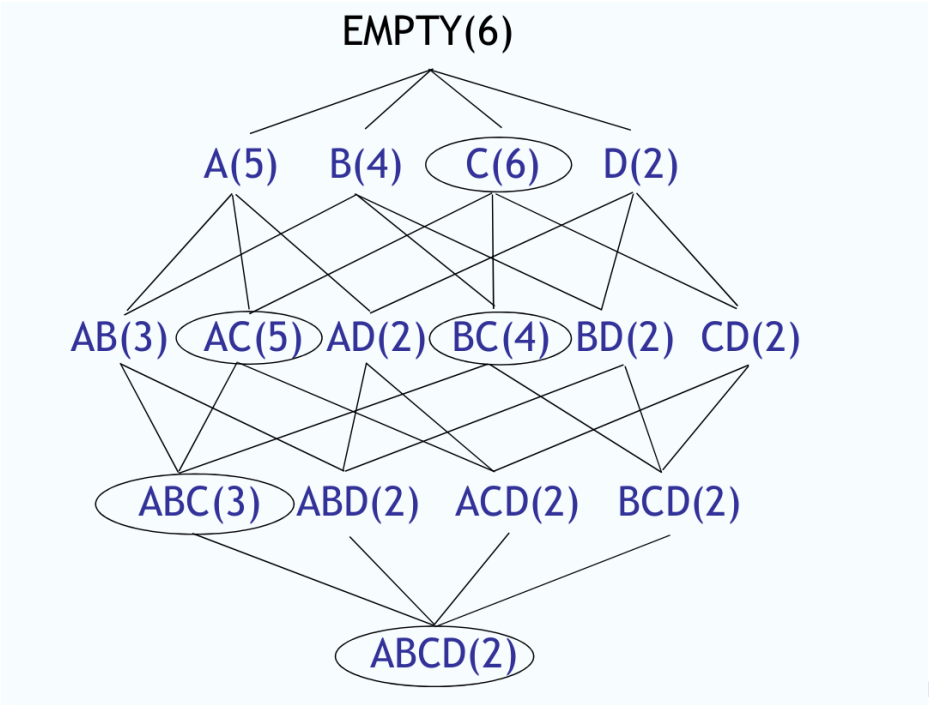
- Neg-Border:

Sets

- Free-Sets: Alle Vorherigen Subsets sind häufiger:
  - Wenn Tupel freq ist, überprüfe Häufigkeit zu seinen Subsets



- Closed-Sets: Alle nachfolgenden Sets müssen weniger häufiger sein
  - Füge am Anfang alle einelementige freq-Tupel zur Lösung hinzu
  - Nach jeder Iteration gehe über die freq-Tupel und lösche die dazugehörigen Subtupel aus der Lösung, die gleich häufig sind



Code

```
def read(file, threshold, frequency):
    file = open(file, "r")
    data = []
    count = None
    for line in file:
        tmp = [int(x) for x in line.strip().split("
        number = 0
        if not count:
            count = [0 for i in range(len(tmp))]
        for i in range(len(tmp)):
            if tmp[i] == 1:
                count[i] += 1
                number += 2 ** i
```

Runtimes ohne Berechnung der Borders und Sets

file \threshold	0.4	0.5	0.6	0.7	0.8	0.9
dm1	5,860	9,220	8,799	7,960	7,270	6,499
dm2	8,300	9,790	9,769	6,669	9,790	6,369
dm3	119,750	9,280	19,299	7,730	10,140	11,339
dm4	23431	5249	1878	859,5	597,7	341,1

Tabelle: alle Zeiten in $*10^{-4}s$

Runtimes mit Berechnung Borders und Sets

file\threshold	0.4	0.5	0.6	0.7	0.8	0.9
dm1	2,636	1,790	1,668	1,492	1,425	1,201
dm2	1,688	1,306	1,153	1,172	1,192	1,157
dm3	11,58	5,757	4,318	2,939	1,916	1,263
dm4	11669	2145	289,903	71,260	31,589	21,732

Tabelle: alle Zeiten in $*10^{-3}s$