Mining Association Rules

——Efficient and scalable frequent itemset mining methods——

徐华

清华大学 计算机系 智能技术与系统国家重点实验室

xuhua@tsinghua.edu.cn

1

Association and Correlations

- Association and Correlations
- Efficient and Scalable Frequent Itemset Mining Methods
- Mining Various Kinds of Association Rules
- From Association Mining to Correlation Analysis
- Constraint-based Association Mining

2

Scalable Methods for Mining Frequent Patterns

- The downward closure (向下闭) property of frequent patterns
  - Any subset of a frequent itemset must be frequent
  - If **{beer, diaper, nuts}** is frequent, so is **{beer, diaper}**
  - i.e., every transaction having {beer, diaper, nuts} also contains {beer, diaper}
- Scalable mining methods: Three major approaches
  - Apriori (Agrawal & Srikant@VLDB'94)
  - Freq. pattern growth (FPgrowth—Han, Pei & Yin @SIGMOD'00)
  - Vertical data format approach (Charm—Zaki & Hsiao @SDM'02)

Apriori: A Candidate Generation-and-Test Approach

- Apriori pruning principle: If there is any itemset which is infrequent, its superset should not be generated/tested! (Agrawal & Srikant @VLDB'94, Mannila, et al. @ KDD'94)
- Method:
  - Initially, scan DB once to get frequent 1-itemset
  - Generate length (k+1) candidate itemsets from length k frequent itemsets
  - Test the candidates against DB
  - Terminate when no frequent or candidate set can be generated

The Apriori Algorithm—An Example

$Sup_{min} = 2$

Database TDB

Tid	Items
10	A, C, D
20	B, C, E
30	A, B, C, E
40	B, E

1^{st} 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

C_2

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

How?

2^{nd} scan

C_2

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

L_2

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

why

C_3

Itemset
{B, C, E}

3^{rd} scan

L_3

Itemset	sup
{B, C, E}	2

The Apriori Algorithm

- **Pseudo-code:**

  C_k: Candidate itemset of size k

  L_k : Frequent itemset of size k

  L_1 = {frequent items};

  for (k = 1; L_k !=∅; k++) **do begin**

  C_{k+1} = candidates generated from L_k;

  for each transaction t in database do

  increment the count of all candidates in C_{k+1}

  that are contained in t

  L_{k+1} = candidates in C_{k+1} with min_support

  end

  return $\cup_k L_k$;

Important Details of Apriori

- How to generate candidates?
  - Step 1: self-joining L_k
  - Step 2: pruning
- How to count supports of candidates?
- Example of Candidate-generation
  - L_3={*abc, abd, acd, ace, bcd*}
  - Self-joining: L_3*L_3
    - *abcd* from *abc* and *abd*
    - *acde* from *acd* and *ace*
  - Pruning:
    - *acde* is removed because *ade* is not in L_3
  - C_4={*abcd*}

7

How to Generate Candidates?

- Suppose the items in L_{k-1} are listed in an order
- Step 1: self-joining L_{k-1}

  insert into C_k

  select $p.item_1, p.item_2, \ldots, p.item_{k-1}, q.item_{k-1}$

  from L_{k-1} p, L_{k-1} q

  where $p.item_1=q.item_1, \ldots, p.item_{k-2}=q.item_{k-2}, p.item_{k-1} < q.item_{k-1}$
- Step 2: pruning

  For all *itemsets c in* C_k **do**

  For all *(k-1)-subsets s of c* **do**

  if ***(s is not in*** L_{k-1}***)*** then delete ***c*** from C_k

8

Challenges of Frequent Pattern Mining

- Challenges
  - Multiple scans of transaction database
  - Huge number of candidates
  - Tedious workload of support counting for candidates
- Improving Apriori: general ideas
  - Reduce passes of transaction database scans
  - Shrink number of candidates
  - Facilitate support counting of candidates

Bottleneck of Frequent-pattern Mining

- Multiple database scans are costly
- Mining long patterns needs many passes of scanning and generates lots of candidates
  - To find frequent itemset $i_1 i_2 \ldots i_{100}$
    - # of scans: 100
    - # of Candidates: $\binom{100}{1} + \binom{100}{2} + \ldots + \binom{100}{100} = 2^{100}-1 = 1.27*10^{30}$!
- Bottleneck: candidate-generation-and-test
- Can we avoid candidate generation?

Mining Frequent Patterns Without Candidate Generation

- Grow long patterns from short ones using local frequent items
  - "abc" is a frequent pattern
  - Get all transactions having "abc" : DB|abc
  - "d" is a local frequent item in DB|abc → abcd is a frequent pattern

11

Construct FP-tree from a Transaction Database

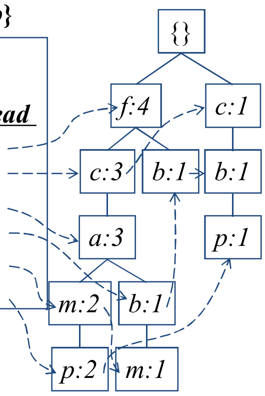
TID	*Items bought*	*(ordered) frequent items*
100	*{f, a, c, d, g, i, m, p}*	*{f, c, a, m, p}*
200	*{a, b, c, f, l, m, o}*	*{f, c, a, b, m}*
300	*{b, f, h, j, o, w}*	*{f, b}*
400	*{b, c, k, s, p}*	*{c, b, p}*
500	*{a, f, c, e, l, p, m, n}*	*{f, c, a, m, p}*

min_support = 3

- Scan DB once, find frequent 1-itemset (single item pattern)
- Sort frequent items in frequency descending order, f-list
- Scan DB again, construct FP-tree

Header Table

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



F-list=f-c-a-b-m-p

12

Benefits of the FP-tree Structure

- Completeness（完备性）
  - Preserve complete information for frequent pattern mining
  - Never break a long pattern of any transaction
- Compactness（紧致性）
  - Reduce irrelevant info—infrequent items are gone
  - Items in frequency descending order: the more frequently occurring, the more likely to be shared
  - Never be larger than the original database (not count node-links and the *count* field)

13

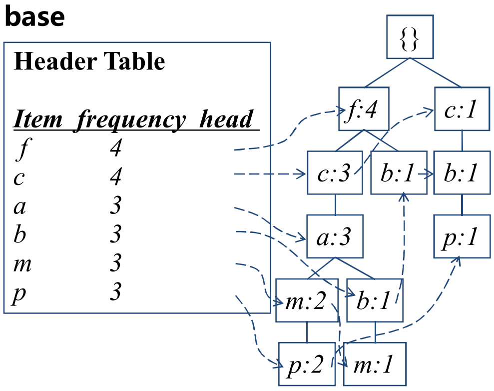
Partition Patterns and Databases

- Frequent patterns can be partitioned into subsets according to f-list
  - F-list=f-c-a-b-m-p
  - Patterns containing p
  - Patterns having m but no p
  - …
  - Patterns having c but no a nor b, m, p
  - Pattern f
- Completeness and non-redundency

14

Find Patterns Having P From P-conditional Database

- Starting at the frequent item header table in the FP-tree
- Traverse the FP-tree by following the link of each frequent item *p*
- Accumulate all of *transformed prefix paths* of item *p* to form *p*′ s conditional pattern base

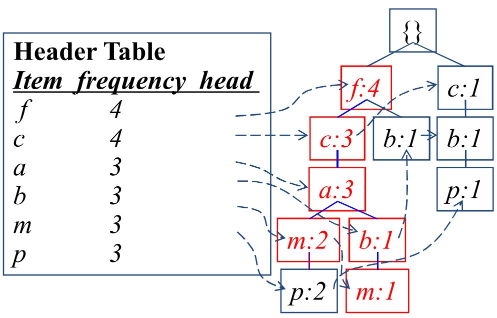


Conditional **pattern bases**

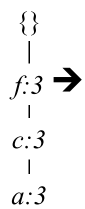
item	*cond. pattern base*
c	*f:3*
a	*fc:3*
b	*fca:1, f:1, c:1*
m	*fca:2, fcab:1*
p	*fcam:2, cb:1*

From Conditional Pattern-bases to Conditional FP-trees

- For each pattern-base
  - Accumulate the count for each item in the base
  - Construct the FP-tree for the frequent items of the pattern base



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

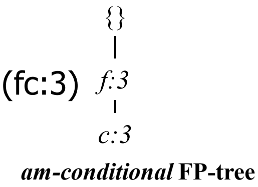


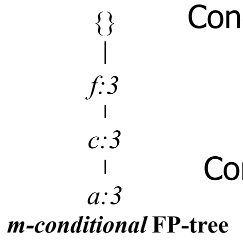
m-conditional **FP-tree**

All frequent patterns relate to *m*
m,
fm, cm, am,
fcm, fam, cam,
fcam

Recursion: Mining Each Conditional FP-tree

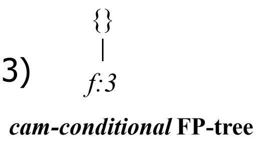
Cond. pattern base of "am": (fc:3)





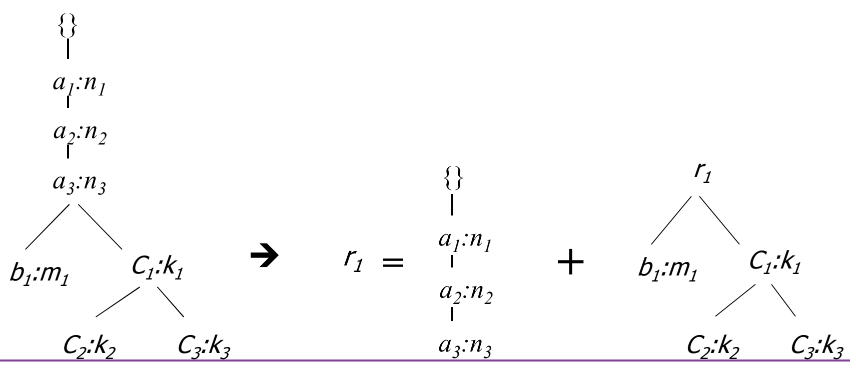
Cond. pattern base of "m": (f:3)

Cond. pattern base of "cam": (f:3)



A Special Case: Single Prefix Path in FP-tree

- Suppose a (conditional) FP-tree T has a shared single prefix-path P
- Mining can be decomposed into two parts
  - Reduction of the single prefix path into one node
  - Concatenation of the mining results of the two parts



Mining Frequent Patterns With FP-trees

- **Idea: Frequent pattern growth**
  - **Recursively grow frequent patterns by pattern and database partition**
- **Method**
  - **For each frequent item, construct its conditional pattern-base, and then its conditional FP-tree**
  - **Repeat the process on each newly created conditional FP-tree**
  - **Until the resulting FP-tree is empty, or it contains only one path—single path will generate all the combinations of its sub-paths, each of which is a frequent pattern**

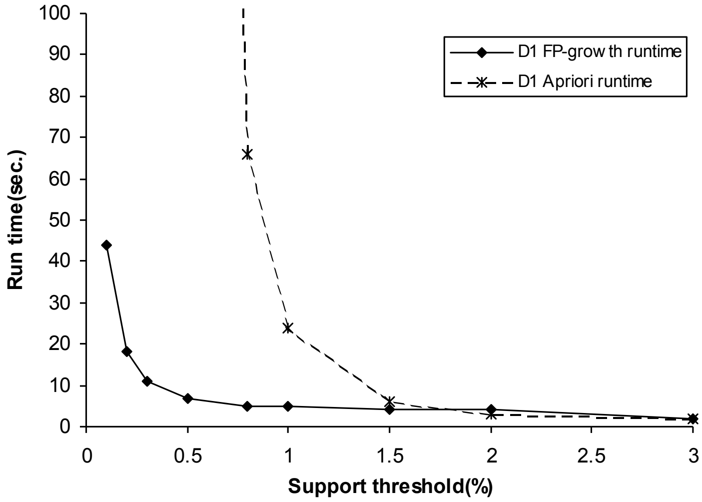
FP-growth Algorithm

<u>FP tree construct procedure:</u>

- **Scan the transaction database D once, construct the F-list**
- **Scan the transaction database D again,**
- **Construct the FP_tree using Insert_tree()**
- **procedure FP_growth(Tree, α)**

(1) if Tree contains a single path P then
(2) for each combination (β) of the nodes in the path P
(3) generate pattern $\beta \cup \alpha$ with support count = minimum support count of nodes in β;
(4) else for each a_i in the header of Tree {
(5) generate pattern =$\beta \cup a_i$ with support count = a_i.support;
(6) construct β's conditional pattern base and then β 's conditional FP tree *Tree* ;}
(7) if $Tree_\beta \neq \phi$; then
(8) call FP_growth($Tree_\beta$; β);

FP-Growth vs. Apriori: Scalability With the Support Threshold



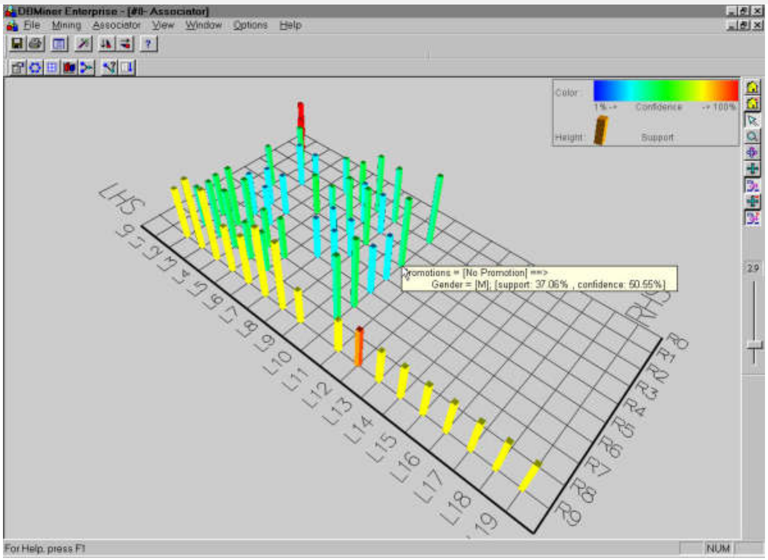
21

Why Is FP-Growth the Winner?

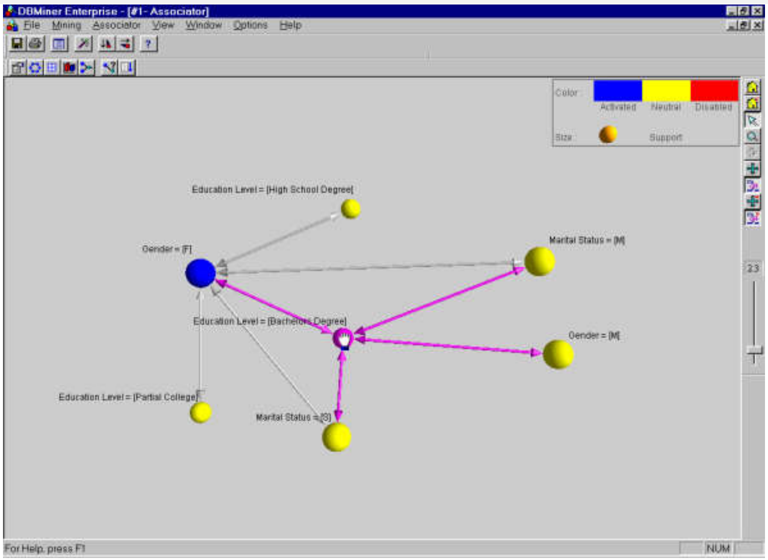
- Divide-and-conquer:
  - decompose both the mining task and DB according to the frequent patterns obtained so far
  - leads to focused search of smaller databases
- Other factors
  - no candidate generation, no candidate test
  - compressed database: FP-tree structure
  - no repeated scan of entire database
  - basic ops—counting local freq items and building sub FP-tree, no pattern search and matching

22

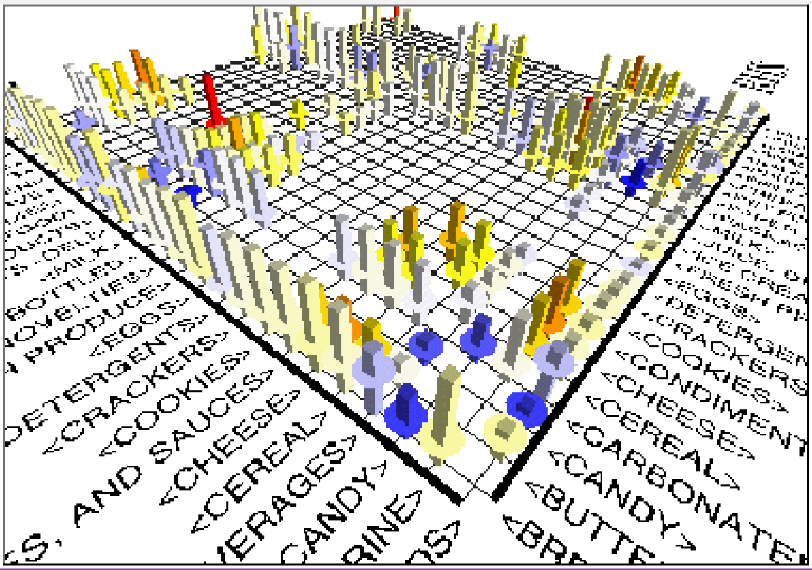
Visualization of Association Rules: Plane Graph



Visualization of Association Rules: Rule Graph



Visualization of Association Rules (SGI/MineSet 3.0)



25

Thanks !

26