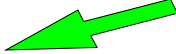


DWDM : Unit IV

Mining Association Rules in Large Databases

UNIT – IV: Mining Association Rules in Large Databases: Association Rule Mining, Mining Single-Dimensional Boolean Association Rules from Transactional Databases, Mining Multilevel Association Rules from Transaction Databases.

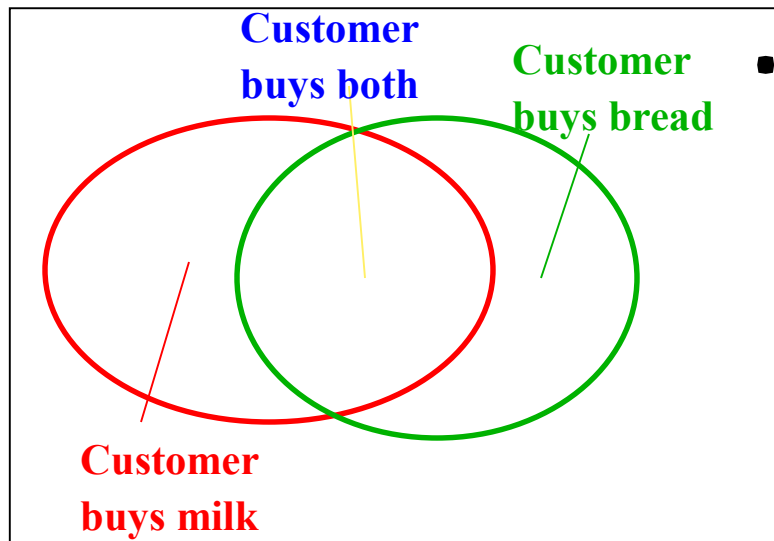
Mining Frequent Patterns, Association and Correlations

- ☐ Association rule mining 
- ☐ Mining single-dimensional Boolean association rules from transactional databases
- ☐ Mining multilevel association rules from transactional databases
- ☐ Summary

What Is Association Mining?

- Association rule mining:
 - Finding frequent patterns, associations, correlations, or causal structures among sets of items or objects in transaction databases, relational databases, and other information repositories.
- Applications:
 - Basket data analysis, cross-marketing, catalog design, loss-leader analysis, clustering, classification, etc.
- Examples.
 - Rule form: "Body \rightarrow Head [support, confidence]".
 - $\text{buys}(x, \text{"milk"}) \rightarrow \text{buys}(x, \text{"bread"}) [0.5\%, 60\%]$
 - $\text{major}(x, \text{"CS"}) \wedge \text{takes}(x, \text{"DB"}) \rightarrow \text{grade}(x, \text{"A"}) [1\%, 75\%]$

Rule Measures: Support and Confidence



- Find all the rules $X \& Y \Rightarrow Z$ with minimum confidence and support
 - support, s , **probability** that a transaction contains $\{X \cup Y \cup Z\}$
 - confidence, c , **conditional probability** that a transaction having $\{X \cup Y\}$ also contains Z

Transaction ID	Items Bought
2000	A,B,C
1000	A,C
4000	A,D
5000	B,E,F

Let minimum support 50%, and minimum confidence 50%, we have

- $A \Rightarrow C$ (50%, 66.6%)
- $C \Rightarrow A$ (50%, 100%)

What Is Frequent Pattern Analysis?

- Frequent pattern: a pattern (a set of items, subsequences, substructures, etc.) that occurs frequently in a data set.
-
- First proposed by Agrawal, Imielinski, and Swami in the context of frequent itemsets and association rule mining.
- **Motivation:** Finding inherent regularities in data
 - What products were often purchased together?— PC and Coke ?!
 - What are the subsequent purchases after buying a PC?

Association rule mining steps

How are association rules mined from large databases?" Its a two-step process:

Step 1: Find all frequent itemsets. By definition, each of these itemsets will occur at least as frequently as a pre-determined **minimum support count**.

Step 2: Generate strong association rules from the frequent itemsets. By definition, these rules must satisfy **minimum support and minimum confidence**.

Association Rule Mining: A Road Map

Association rules can be classified in various ways, based on the following criteria:

1. Based on the types of values handled in the rule
 - Boolean vs. quantitative associations
2. Based on the dimensions of data involved in the rule
 - Single dimension vs. multiple dimensional associations
3. Based on the levels of abstractions involved in the rule set
 - Single level vs. multiple-level analysis
4. Based on the nature of the association involved in the rule
 - Various extensions- Correlation analysis, Max Patterns generation, Frequent Itemset Counting

- Boolean vs. quantitative associations (Based on the **types of values handled**)
 - $\text{buys}(x, \text{"SQLServer"}) \wedge \text{buys}(x, \text{"DMBook"}) \rightarrow \text{buys}(x, \text{"DBMiner"})$ [0.2%, 60%]
 - $\text{age}(x, \text{"30..39"}) \wedge \text{income}(x, \text{"42..48K"}) \rightarrow \text{buys}(x, \text{"Macbook"})$ [1%, 75%]
- Single dimension vs. multiple dimensional associations (see ex. Above)
- Single level vs. multiple-level analysis
 - What brands of beers are associated with what brands of noodles?
- Various extensions
 - Correlation, causality analysis
 - Association does not necessarily imply correlation or causality
 - Maxpatterns and closed itemsets
 - Constraints enforced
 - E.g., small sales ($\text{sum} < 100$) trigger big buys ($\text{sum} > 1,000$)?

Mining Frequent Patterns, Association and Correlations

- ❑ Association rule mining
- ❑ Mining single-dimensional Boolean association rules from transactional databases
- ❑ Mining multilevel association rules from transactional databases
- ❑ Summary

Scalable Methods for Mining Frequent Patterns

The downward closure property of frequent patterns

- Any subset of a frequent itemset must be frequent
- If {Coke, Bread, Nuts} is frequent, so is {Coke, Bread}
- i.e., every transaction having {Coke, Bread, Nuts} also contains {Coke, Bread}

Scalable mining methods: Three major approaches

1. Apriori
2. Freq. Pattern growth
3. Vertical data format approach

Mining Association Rules—An Example

Transaction ID	Items Bought
2000	A,B,C
1000	A,C
4000	A,D
5000	B,E,F

Min. support 50%
Min. confidence 50%

Frequent Itemset	Support
{A}	75%
{B}	50%
{C}	50%
{A,C}	50%

For rule $A \Rightarrow C$:

support = support($\{A \cup C\}$) = 50%

confidence = support($\{A \cup C\}$)/support($\{A\}$) = 66.6%

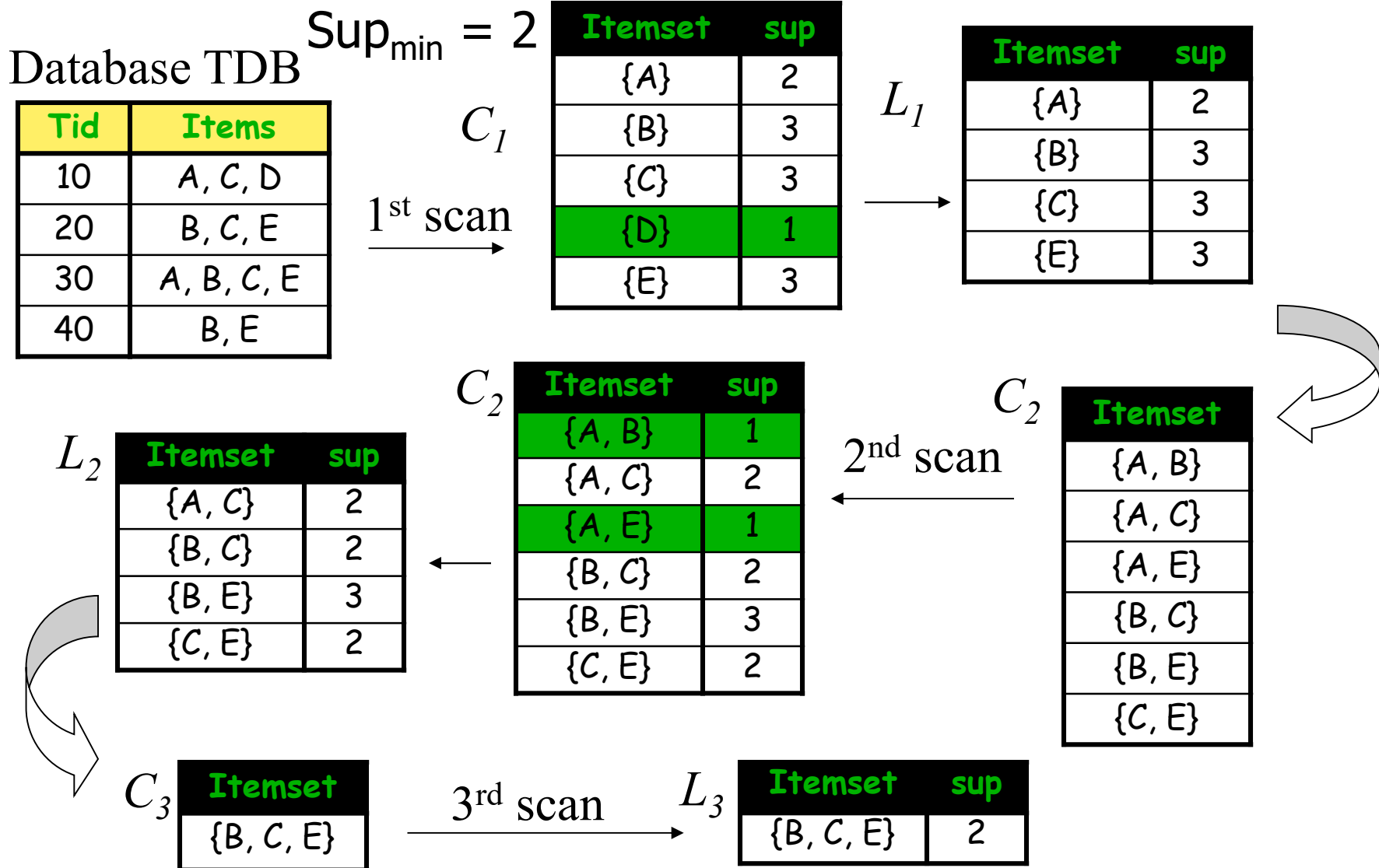
The **Apriori** principle:

Any **subset** of a frequent itemset must be frequent

Mining Frequent Itemsets: the Key Step

- Find the *frequent itemsets*: the sets of items that have minimum support
 - A subset of a frequent itemset must also be a frequent itemset
 - i.e., if $\{AB\}$ is a frequent itemset, both $\{A\}$ and $\{B\}$ should be a frequent itemset
 - Iteratively find frequent itemsets with cardinality from 1 to k (k -itemset)
- Use the frequent itemsets to generate association rules.

The Apriori Algorithm—An Example



The Apriori Algorithm

- **Join Step:** C_k is generated by joining L_{k-1} with itself
- **Prune Step:** Any $(k-1)$ -itemset that is not frequent cannot be a subset of a frequent k -itemset

- Pseudo-code:

C_k : Candidate itemset of size k

L_k : frequent itemset of size k

$L_1 = \{\text{frequent items}\};$

for ($k = 1; L_k \neq \emptyset; k++$) **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}
contained in t

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

end

return $\cup_k L_k$;

that are

Algorithm Apriori (in Detail)

Find frequent itemsets using an iterative level-wise approach.

Input: Database, D , of transactions; minimum support threshold, \min_sup .

Output: L , frequent itemsets in D . (L is after pruning wrt \min_sup)

Method:

- 1) $L_1 = \text{find_frequent_1-itemsets}(D);$
- 2) for ($k = 2; L_{k-1} \neq \Phi; k++$) {
- 3) $C_k = \text{apriori_gen}(L_{k-1}, \min_sup);$
- 4) for each transaction $t \in D$ { //scan D for counts
- 5) $C_t = \text{subset}(C_k, t);$ //get subsets of t that are candidates
- 6) for each candidate $c \in C_t$
- 7) $c.\text{count}++;$
- 8) }
- 9) $L_k = \{c \in C_k \mid c.\text{count} \geq \min_sup\}$
- 10) }
- 11) return $L = \bigcup_k L_k$

Procedures: apriori_gen and has_infrequent_subset

```
procedure apriori_gen( $L_{k-1}$ : frequent  $(k-1)$ -itemsets;  $min\_sup$ : minimum support)
1)  for each itemset  $l_1 \in L_{k-1}$ 
2)    for each itemset  $l_2 \in L_{k-1}$ 
3)      if  $(l_1[1] = l_2[1]) \wedge (l_1[2] = l_2[2]) \wedge \dots \wedge (l_1[k-2] = l_2[k-2]) \wedge (l_1[k-1] < l_2[k-1])$  then {
4)         $c = l_1 \bowtie l_2$ ; // join step: generate candidates
5)        if has_infrequent_subset( $c, L_{k-1}$ ) then
6)          delete  $c$ ; // prune step: remove unfruitful candidate
7)        else add  $c$  to  $C_k$ ;
8)      }
9)  return  $C_k$ ;
```

```
procedure has_infrequent_subset( $c$ : candidate  $k$ -itemset;  $L_{k-1}$ : frequent  $(k-1)$ -itemsets); // use prior knowledge
1)  for each  $(k-1)$ -subset  $s$  of  $c$ 
2)    if  $s \notin L_{k-1}$  then
3)      return TRUE;
4)  return FALSE;
```


Generating association rules from frequent itemsets

Generating association rules from frequent itemsets is straightforward.

- Strong association rules satisfy both minimum support and minimum confidence

$$\text{confidence}(A \Rightarrow B) = \text{Prob}(B/A) = \text{support}(A \cup B) / \text{support}(A)$$

Based on this equation, association rules can be generated as follows.

- For each frequent itemset, I , generate all non-empty subsets of I . (Ex $\{B, C, E\} - \{B\}, \{C\}, \{E\}, \{B, C\}, \{C, E\}, \{B, E\}, \{B, C, E\}$)
- For every non-empty subset s , of I , output the rule

$$s \Rightarrow (I - s) \text{ if } \text{support}(I) / \text{support}(s) \geq \text{min_conf.}$$

where min conf is the minimum confidence threshold.

Suppose the data contains the frequent itemset $I = \{B, C, E\}$. The association rules that can be generated from I :

The non-empty subsets of I are $\{B, C\}$, $\{B, E\}$, $\{C, E\}$, $\{B\}$, $\{C\}$, and $\{E\}$.

The resulting association rules are as shown below, each listed with its confidence.

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

$B \wedge C \Rightarrow E$, confidence = $2/2 = 100\%$

$B \wedge E \Rightarrow C$, confidence = $2/3 = 67\%$

$C \wedge E \Rightarrow B$, confidence = $2/2 = 100\%$

$B \Rightarrow C \wedge E$, confidence = $2/3 = 67\%$

$C \Rightarrow B \wedge E$, confidence = $2/3 = 67\%$

$E \Rightarrow B \wedge C$, confidence = $2/3 = 67\%$

If the minimum confidence threshold is, say, 70%, then only the **first and third rules above are output**, since these are the only ones generated that are strong.

Methods to Improve Apriori's Efficiency

Also called Variations of the Apriori algorithm

1. **Hash-based itemset counting:** A k -itemset whose corresponding hashing bucket count is below the threshold cannot be frequent
2. **Transaction reduction:** A transaction that does not contain any frequent k -itemset is useless in subsequent scans
3. **Partitioning:** Any itemset that is potentially frequent in DB must be frequent in at least one of the partitions of DB
4. **Sampling:** mining on a subset of given data, lower support threshold + a method to determine the completeness
5. **Dynamic itemset counting:** add new candidate itemsets only when all of their subsets are estimated to be frequent

Hash-based technique

- The basic idea in hash coding is to determine the address of the stored item as some simple arithmetic function content
- Map onto a subspace of allocated addresses using a hash function
- Assume the allocated address range from b to $n+b-1$, the hashing function may take $h=(a \bmod n)+b$
- In order to create a good pseudorandom number, n ought to be prime

Transactional DB→

Hash based Itemset Counting

Hash table with hash function

$$h(x, y) = ((\text{order of } x)10 + (\text{order of } y)) \bmod 7$$

TID	List of item IDs
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

bucket address	0	1	2	3	4	5	6
bucket count	2	2	4	2	2	4	4
bucket contents	{I1, I4} {I3, I5}	{I1, I5} {I1, I5}	{I2, I3} {I2, I3} {I2, I3} {I2, I3}	{I2, I4} {I2, I4}	{I2, I5} {I2, I5}	{I1, I2} {I1, I2} {I1, I2} {I1, I2}	{I1, I3} {I1, I3} {I1, I3} {I1, I3}

Hash table, *H2*, for candidate 2-itemsets. This hash table was generated by scanning transactions while determining *L1*. If the minimum support count is, say, 3, then the itemsets in buckets 0, 1, 3, and 4 cannot be frequent and so they should not be included in *C2*.

Hash-based technique (contd.)

- Two different keywords may have equal hash addresses.
- Partition the memory into buckets, and to address each bucket
 - One address is mapped into one bucket
- When scanning each transition in the database to generate frequent 1-itemsets, we can generate all the 2-itemsets for each transition and hash them into different buckets of the hash table.
- We use $h = a \bmod n$, a address, $n < \text{the size of } C_2$.
- A 2-itemset whose bucket count in the hash table is below the support threshold cannot be frequent, and should be removed from the candidate set

Transaction reduction

- A transaction which does not contain frequent k-itemsets should be removed from the database for further scans

Partitioning

- First scan:
 - Subdivide the transactions of database D into n non overlapping partitions
 - If the minimum support in D is min_sup , then the minimum support for a partition is $min_sup * \text{number of transactions in that partition}$
 - Local frequent items are determined
 - A local frequent item may not be a frequent item in D
- Second scan:
 - Frequent items are determined from the local frequent items

Sampling

- Pick a random sample S of D
- Search for local frequent items in S
 - Use a lower support threshold
 - Determine frequent items from the local frequent items
 - Frequent items of D may be missed
- For completeness a second scan is done

Dynamic Itemset Counting: Reduce Number of Scans

- adding candidate itemsets at different points during a scan.
- Database is partitioned into blocks marked by start points
- new candidate itemsets can be added at any start point, unlike in Apriori,
- The technique uses the count-so-far as the lower bound of the actual count. If the count-so-far \geq min_sup, the itemset is added into the frequent itemset collection

Bottlenecks of Apriori Algorithm

Apriori candidate generate-and-test method significantly reduces the size of candidate sets, leading to good performance gain.

However, it can suffer from two nontrivial costs:

1. *It may need to generate a huge number of candidate sets. : Multiple database scans are costly.*
 - To find frequent itemset $i_1 i_2 \dots i_{100}$
 - # of scans: 100
 - # of Candidates: $\binom{100}{1} + \binom{100}{2} + \dots + \binom{100}{100} = 2^{100} - 1 = 1.27 \times 10^{30}!$
2. *Mining long patterns needs many passes of scanning and generates lots of candidates*

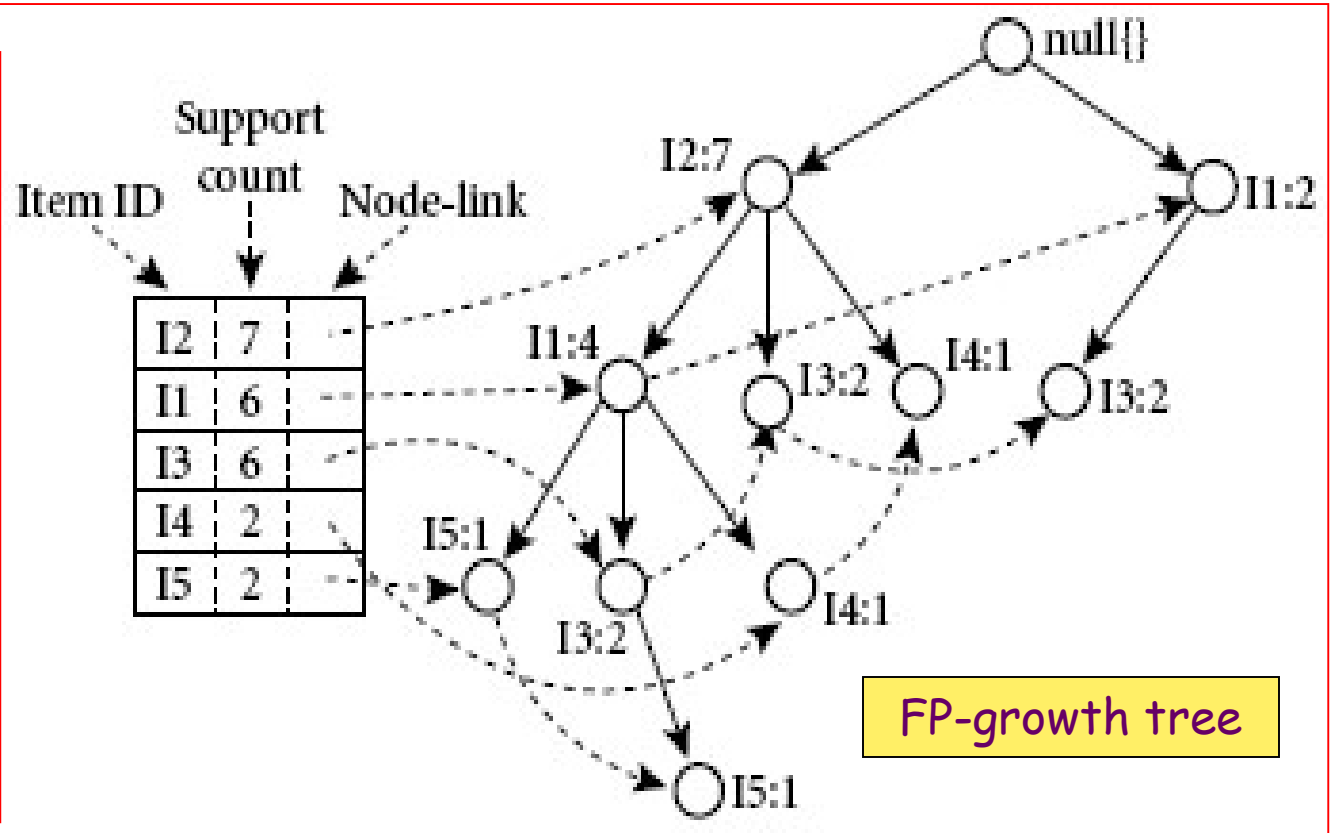
Mining Frequent Patterns Without Candidate Generation

- *Mines the complete set of frequent itemsets without candidate generation.*
- Frequent-pattern growth (or FP-growth):- adopts a *divide-and-conquer* strategy.
- First, it compresses the database representing frequent items into a frequent-pattern tree (FP-tree).

FP-growth tree

Transactional data

<i>TID</i>	<i>List of item JDs</i>
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3



Steps:

1: derive the set of frequent items (1-itemsets) and their support counts (frequencies). **Min_support_count=2.**

2. Sort them in order of descending support count.

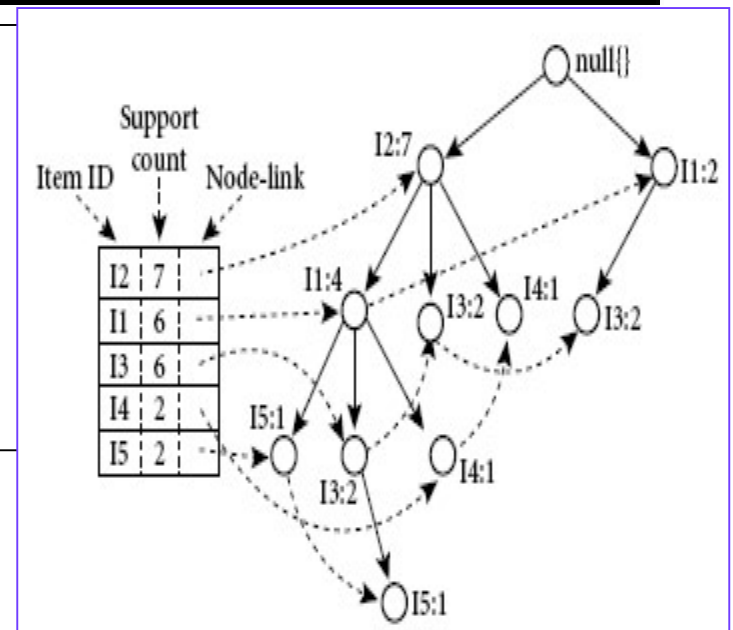
Thus, we have $L = I_2:7, I_1:6, I_3:6, I_4:2, I_5:2$.

Mining the FP-tree by creating conditional (sub-)pattern bases.

Item	Conditional Pattern Base	Conditional FP-tree	Frequent Patterns Generated
I5	$\{\{I2, I1: 1\}, \{I2, I1, I3: 1\}\}$	$\langle I2: 2, I1: 2 \rangle$	$\{I2, I5: 2\}, \{I1, I5: 2\}, \{I2, I1, I5: 2\}$
I4	$\{\{I2, I1: 1\}, \{I2: 1\}\}$	$\langle I2: 2 \rangle$	$\{I2, I4: 2\}$
I3	$\{\{I2, I1: 2\}, \{I2: 2\}, \{I1: 2\}\}$	$\langle I2: 4, I1: 2 \rangle, \langle I1: 2 \rangle$	$\{I2, I3: 4\}, \{I1, I3: 4\}, \{I2, I1, I3: 2\}$
I1	$\{\{I2: 4\}\}$	$\langle I2: 4 \rangle$	$\{I2, I1: 4\}$

Steps:

1. Start from each frequent length-1 pattern (initial suffix pattern), construct its **conditional pattern base** (a "subdatabase," which consists of the set of *prefix paths* in the FP-tree co-occurring with the suffix pattern), then construct its (conditional) FP-tree

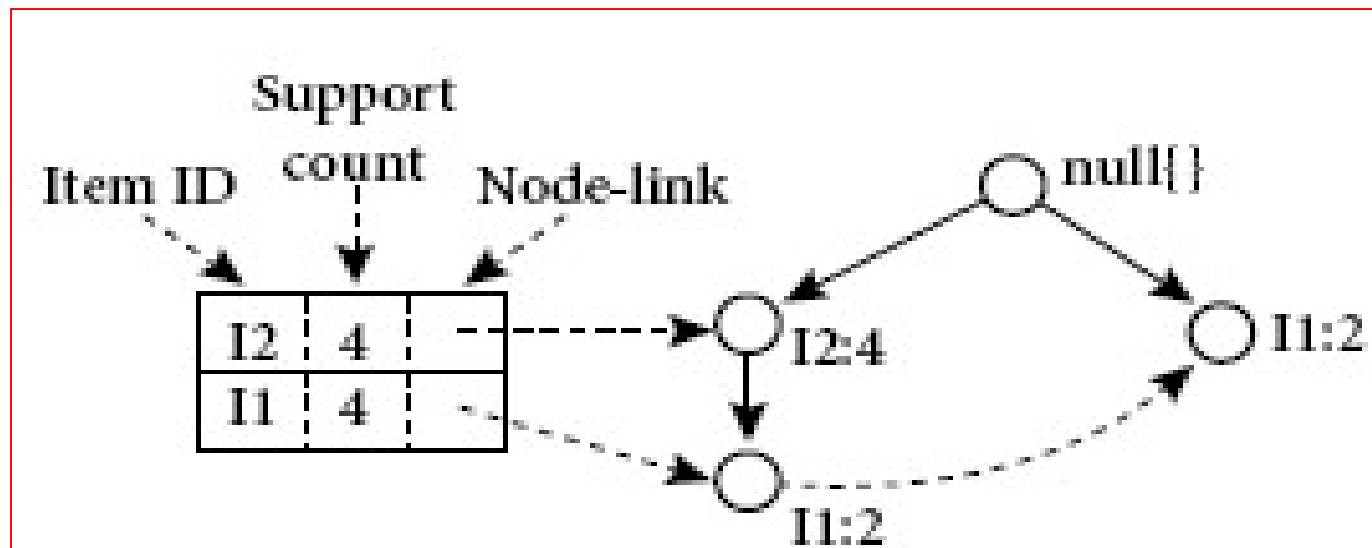


How to find conditional FP tree for I3

Item	Conditional Pattern Base	Conditional FP-tree	Frequent Patterns Generated
I5	$\{\{I2, I1: 1\}, \{I2, I1, I3: 1\}\}$	$\langle I2: 2, I1: 2 \rangle$	$\{I2, I5: 2\}, \{I1, I5: 2\}, \{I2, I1, I5: 2\}$
I4	$\{\{I2, I1: 1\}, \{I2: 1\}\}$	$\langle I2: 2 \rangle$	$\{I2, I4: 2\}$
I3	$\{\{I2, I1: 2\}, \{I2: 2\}, \{I1: 2\}\}$	$\langle I2: 4, I1: 2 \rangle, \langle I1: 2 \rangle$	$\{I2, I3: 4\}, \{I1, I3: 4\}, \{I2, I1, I3: 2\}$
I1	$\{\{I2: 4\}\}$	$\langle I2: 4 \rangle$	$\{I2, I1: 4\}$

I3's conditional pattern base is **I2**, **I1: 2**, **I2: 2**, **I1: 2**.

Its conditional FP-tree has two branches, **I2: 4, I1: 2** and **I1: 2**



Benefits of the FP-tree Structure

- Completeness
 - Preserve complete information for frequent pattern mining
 - Never break a long pattern of any transaction
- Compactness
 - 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)

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

It allows frequent itemset discovery without candidate itemset generation.

Two step approach:

Step 1: Build a compact data structure called the FP-tree

Built using 2 passes over the data-set.

Step 2: Extracts frequent itemsets directly from the FP-tree

Traversal through FP-Tree

FP-Tree is constructed using 2 passes over the data-set

Pass 1:

1. Scan data and find support for each item.
2. Discard infrequent items.
3. Sort frequent items in decreasing order based on their support.
Use this order when building the FP-Tree, so common prefixes can be shared.

Step 1: FP-Tree Construction (Example)

Pass 2: construct the FP-Tree

Read transactions one at a time.

Create nodes for each item of this transaction.

Set counts of each item (i.e., node) to 1.

Read next transaction. Note that if transaction 1 and 2 share an item , the paths may be disjoint as they don't share a common prefix.

Add the link between the item.

If it shares common prefix item, say a, with transaction 1 so the path for transaction 1 and 3 will overlap and the frequency count for node a will be incremented by 1.

Continue until all transactions are mapped to a path in the FP-tree.

1. The FP-tree construction :

(a) Scan the transaction database D once.

- Collect frequent items, and their support counts.
- Sort *list* of frequent items in support count descending order:
L

(b) Create the root of an FP-tree, and label it as “null.” For each transaction *Trans* in D do the following.

- Select and sort the frequent items in *Trans* according to the order of L .

2. The FP-tree Mining

procedure FP growth($Tree, a$)

- (1) if $Tree$ contains a single path P then
- (2) for each combination (denoted as 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 ai in the header of $Tree$ f
- (5) generate pattern $\beta = ai \cup \alpha$ with *support* = $ai.support$;
- (6) construct β 's conditional pattern base and then β 's conditional FP tree $Tree_\beta$;
- (7) if $Tree_\beta \neq \Phi$ then
- (8) call FP growth($Tree_\beta, \beta$); g

Advantages of FP-Growth

- 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

Iceberg Queries

- **Iceberg query:** Compute aggregates over one or a set of attributes only for those whose aggregate values is above certain threshold

- Example:

```
select P.custID, P.itemID, sum(P.qty)
from purchase P
group by P.custID, P.itemID
having sum(P.qty) >= 10
```

- **Compute** iceberg queries efficiently **by Apriori**:
 - First compute lower dimensions
 - Then compute higher dimensions only when **all** the lower ones are above the threshold

Closed Patterns and Max-Patterns

A long pattern contains a combinatorial number of sub-patterns

Solution: Mine *closed patterns* and *max-patterns* instead

- An itemset X is **closed** if X is *frequent* and there exists *no* super-pattern $Y \supset X$, with the same support as X :
- **A closed pattern** -> frequent pattern (meets min_sup criteria). Also, all super-patterns of a closed pattern are less frequent than the closed pattern.

Min_sup
:2

Ex: a) Items: a,b,c
Patterns: {a,b}: 2, {a,b,c}:2
→ {a,b} is frequent but not closed.

3 Items: x,y,z
Patterns: {x,y}:3, {xyz}:2
→ {xy} is a closed pattern

$\text{Sup_Count}(\text{subset}) > \text{Sup_Count}(\text{Superset}) \rightarrow \text{subset is closed} \mid \text{both frequent}$

Closed Patterns and Max-Patterns

- An itemset X is a **max-pattern** if X is frequent and there exists no frequent super-pattern $Y \supset X$.
- A max pattern \rightarrow frequent pattern (meets min_sup criteria like closed pattern). But, Unlike closed pattern, all super-patterns of a max pattern are NOT frequent patterns.

**Min_sup
:2**

Ex: a) Items: a,b,c

Patterns: {a,b}: 3, {a,b,c}:2

\rightarrow {a,b} is frequent but not Max Pattern.

3 Items: x,y,z

Patterns: {x,y}:3, {xyz}:1

\rightarrow {xy} is a Max pattern

**Sup_Count(subset) > Sup_Count(Superset) \rightarrow subset is max pattern |
subset is frequent
superset is NOT frequent**

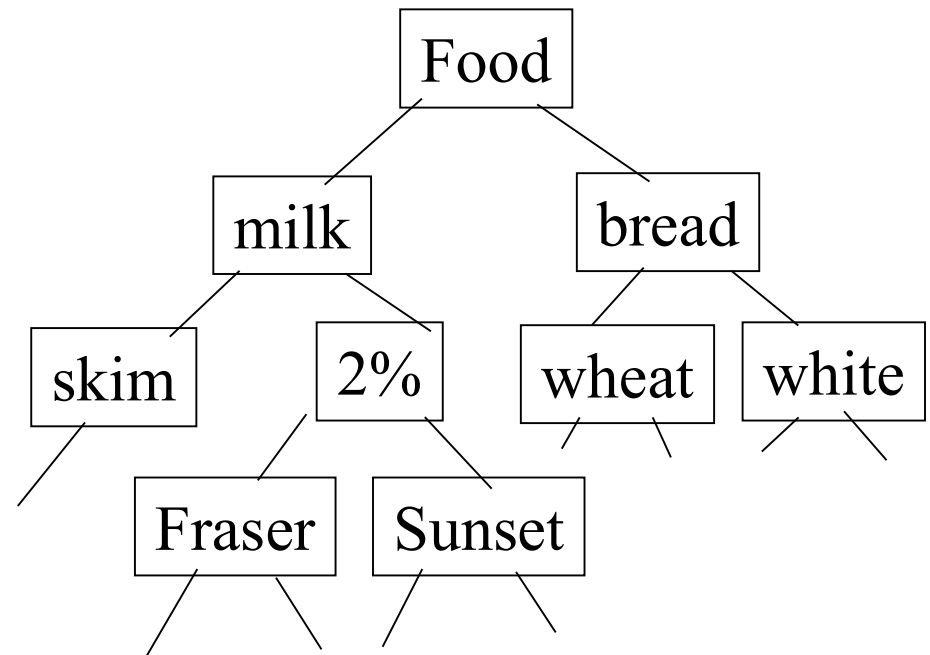
Sup_Count(subset) > Sup_Count(Superset) \rightarrow subset is closed | both frequent

Mining Association Rules in Large Databases

- Association rule mining
- Mining single-dimensional Boolean association rules from transactional databases
- Mining multilevel association rules from transactional databases
- Summary

Multiple-Level Association Rules

- Items often form hierarchy.
(Concept Hierarchy)
- Items at the lower level are expected to have lower support.
- Rules regarding itemsets at appropriate levels could be quite useful.
- Transaction database can be encoded based on dimensions and levels
- We can explore shared multi-level mining →



TID	Items
T1	{111, 121, 211, 221}
T2	{111, 211, 222, 323}
T3	{112, 122, 221, 411}
T4	{111, 121}
T5	{111, 122, 211, 221, 413}

Approaches to Mining Multi-level ARs

Uniform Support vs. Reduced Support

- **Uniform Support: the same minimum support for all levels**
 - + One minimum support threshold. No need to examine itemsets containing any item whose ancestors do not have minimum support.
 - - Lower level items do not occur as frequently. If support threshold
 - too high \Rightarrow miss low level associations
 - too low \Rightarrow generate too many high level associations
- **Reduced Support: reduced minimum support at lower levels**
 - There are 4 search strategies:
 1. Level-by-level independent
 2. Level-cross filtering by single item
 3. Level-cross filtering by k-itemset
 4. Controlled level-cross filtering by single item

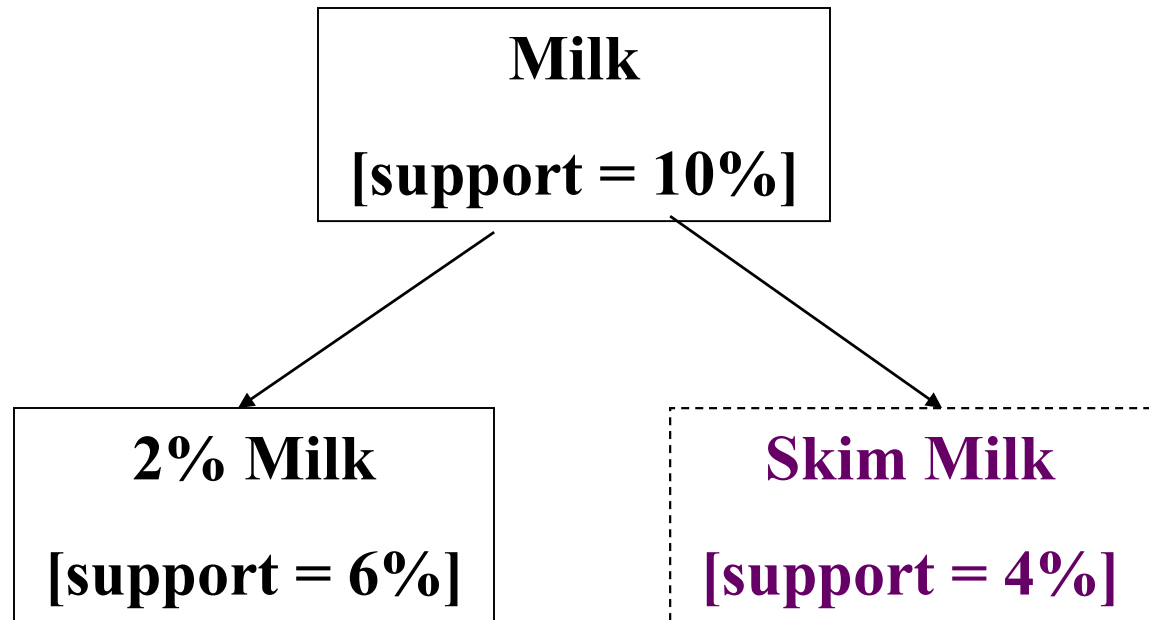
Uniform Support

1. Level-by-level independent (Relaxed)

Multi-level mining with uniform support

Level 1
min_sup = 5%

Level 2
min_sup = 5%

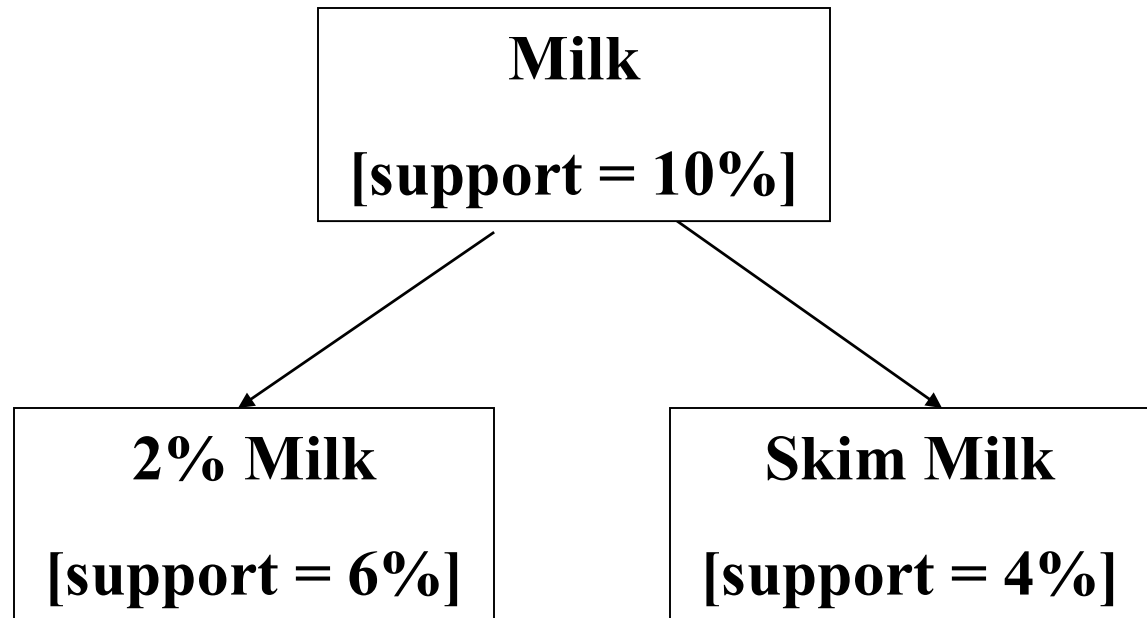


Reduced Support

Multi-level mining with reduced support

Level 1
min_sup = 5%

Level 2
min_sup = 3%



[Back](#)

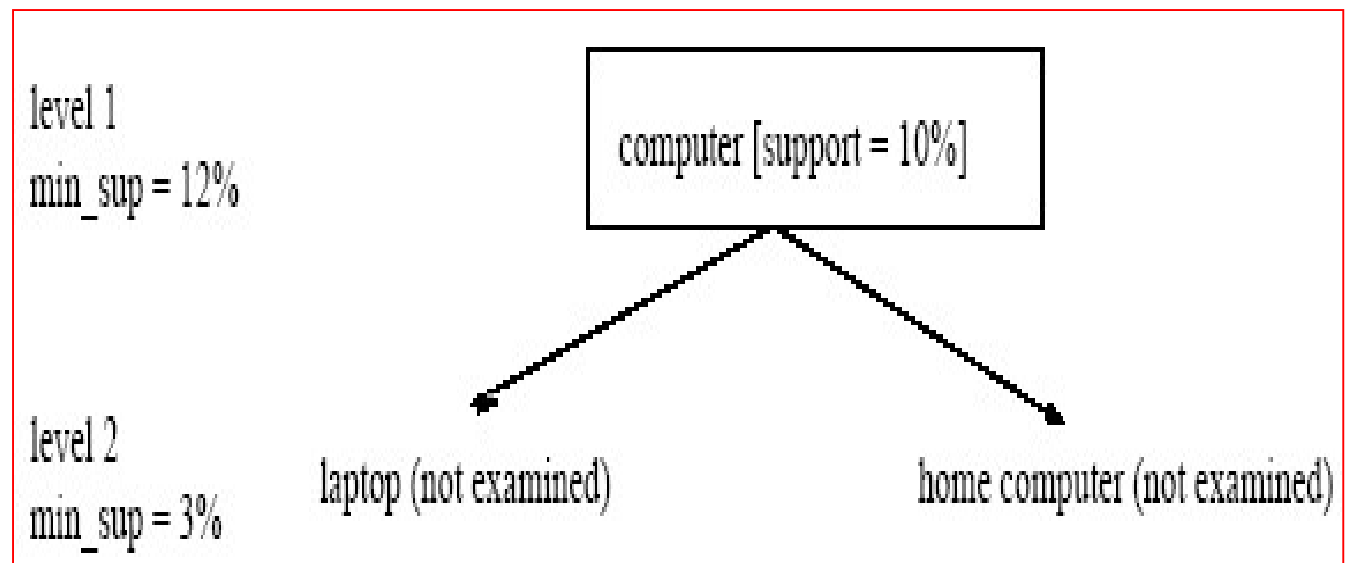
Reduced Support:

- Each node is examined, regardless of whether or not its parent node is found to be frequent.

2. level-cross filtering by single item (Conditional Compromise between 2 extremes)

- An item at the i -th level is examined **iff** its parent node at the $(i-1)$ th level is frequent. Sets a **level** passage threshold for every **level**. Allows the inspection of lower abstractions even if its ancestor fails to satisfy min_sup threshold.

• If a node is frequent, its children will be examined; otherwise, its descendents are pruned from the search.



3. level-cross filtering by k-itemset (Very Strong)

A k-itemset at the i-th level is examined iff its corresponding parent k-itemset at the (i-1)th level is frequent.

- Ex: the 2-itemset {computer, printer} is frequent
- Therefore the nodes {laptop computer, b/w printer}, {laptop computer, color printer}, {home computer, b/w printer} and {home computer, color printer} are examined.

level 1

min_sup = 5%

computer & printer [support = 7%]

level 2

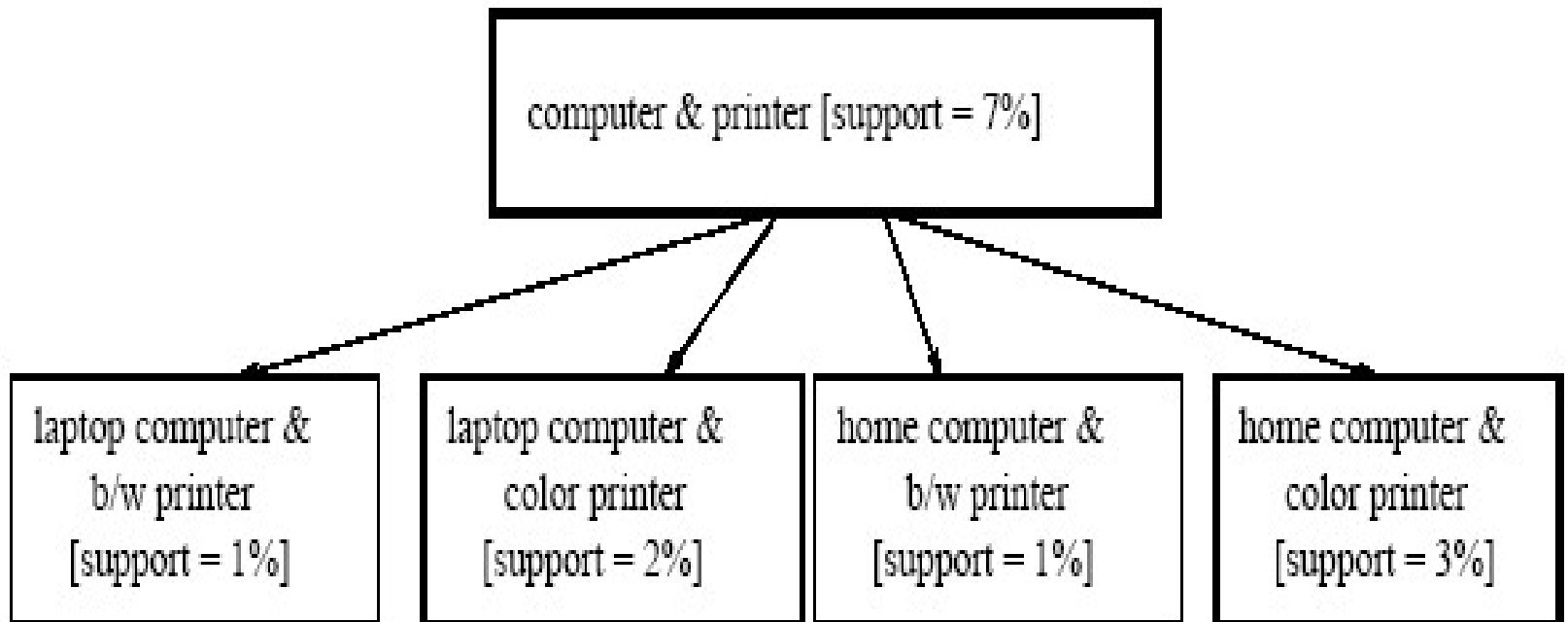
min_sup = 2%

laptop computer &
b/w printer
[support = 1%]

laptop computer &
color printer
[support = 2%]

home computer &
b/w printer
[support = 1%]

home computer &
color printer
[support = 3%]



methods comparision

level-by-level independent : very relaxed

- it may lead to examining numerous infrequent items at low levels,

level-cross filtering by k-itemset : Very Strong

- It allows the mining system to examine only the children of frequent k-itemsets. This restriction is very strong in that there usually are not many k-itemsets (especially when $k > 2$) which, when combined, are also frequent. Hence, many valuable patterns may be filtered out using this approach.

level-cross filtering by single item : compromise between the two extremes.

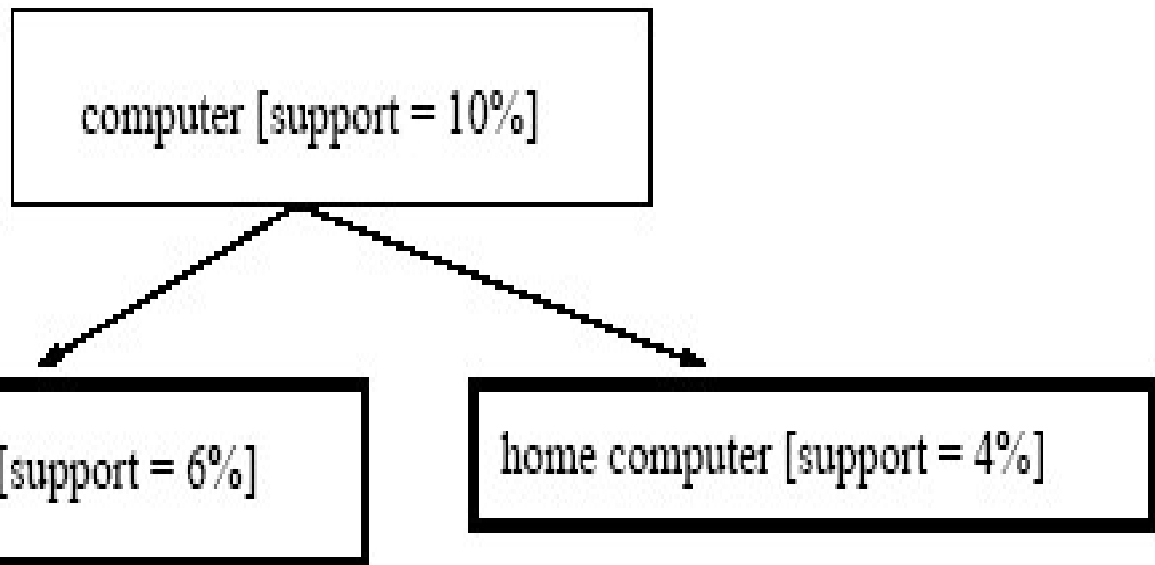
- But this may miss associations between low level items that are frequent based on a reduced min_sup, but whose ancestors do not satisfy minimum support (since the support thresholds at each level can be different).

4. Controlled level-cross filtering by single item strategy

- A threshold, called the **level passage threshold**, can be set up for passing down relatively frequent items (called subfrequent items) to lower levels.
 - i.e., this method allows the **children of items that do not satisfy the minimum support threshold to be examined** if these items satisfy the level passage threshold.
-

level 1
min_sup = 12%
level_passage_sup = 8%

computer [support = 10%]



```
graph TD; A[computer [support = 10%]] --> B[laptop computer [support = 6%]]; A --> C[home computer [support = 4%]]
```

level 2
min_sup = 3%

laptop computer [support = 6%]

home computer [support = 4%]

Multi-level Association: Redundancy Filtering

- Some rules may be redundant due to “ancestor” relationships between items.
- Example
 - milk \Rightarrow wheat bread [support = 8%, confidence = 70%]
 - 2% milk \Rightarrow wheat bread [support = 2%, confidence = 72%]
- We say the first rule is an ancestor of the second rule.
- A rule is redundant if its support is close to the “expected” value, based on the rule’s ancestor.

Mining Association Rules in Large Databases

- Association rule mining
- Mining single-dimensional Boolean association rules from transactional databases
- Mining multilevel association rules from transactional databases
- Mining multidimensional association rules from transactional databases and data warehouse
- From association mining to correlation analysis
- Constraint-based association mining
- Summary

Examples for Practice

Construct FP-tree from a Transaction Database

<i>TID</i>	<i>Items bought</i>	<i>(ordered) frequent items</i>
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}

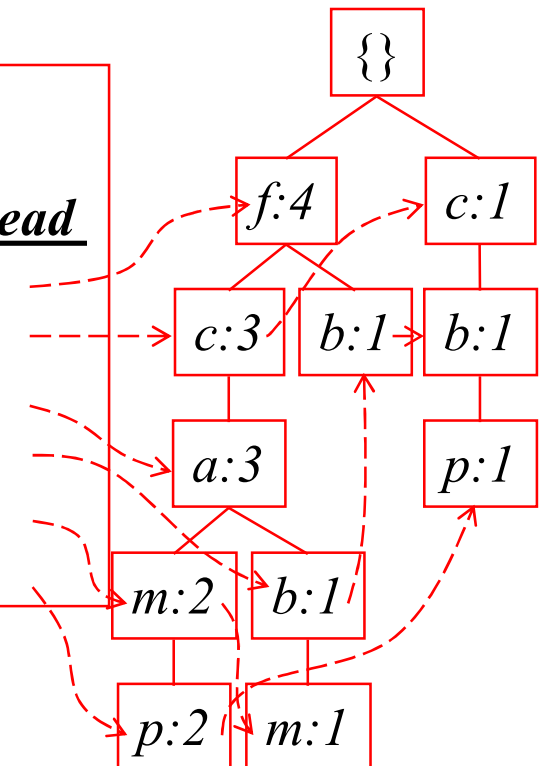
min_support = 3

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

Header Table

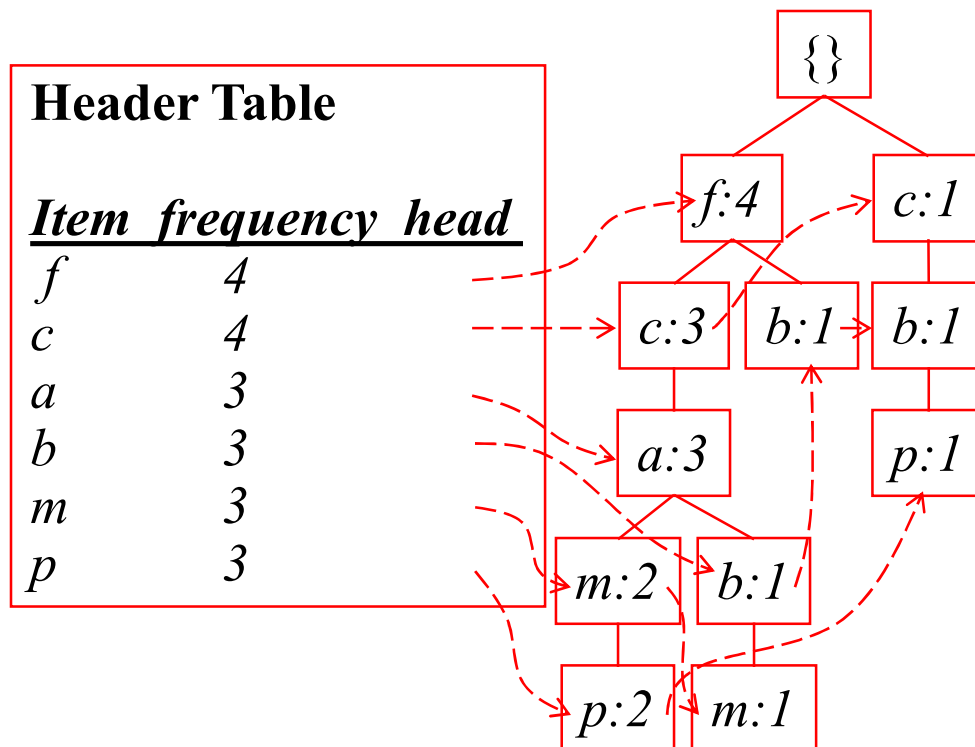
<i>Item</i>	<i>frequency</i>	<i>head</i>
<i>f</i>	4	
<i>c</i>	4	
<i>a</i>	3	
<i>b</i>	3	
<i>m</i>	3	
<i>p</i>	3	

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



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

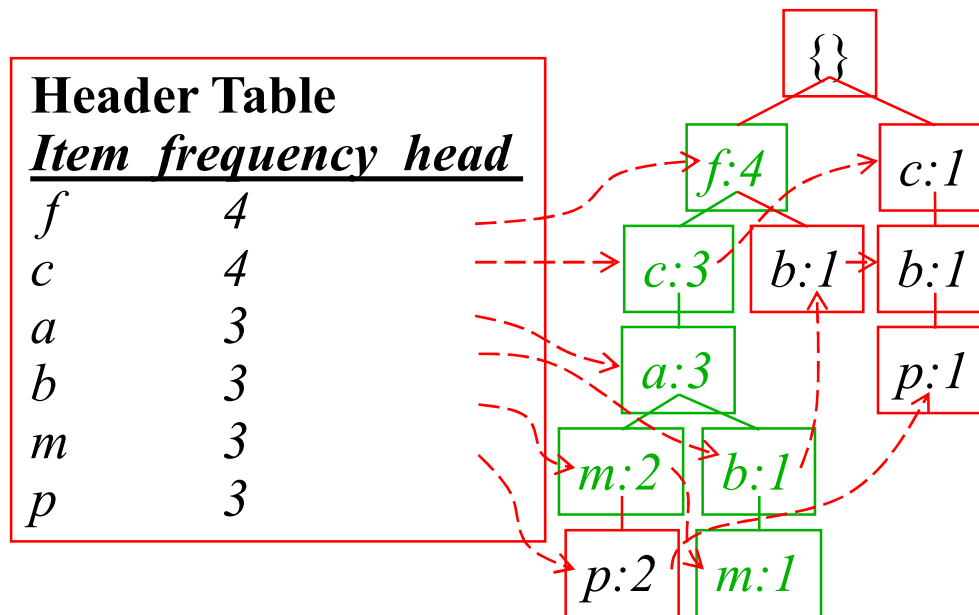


Conditional pattern bases

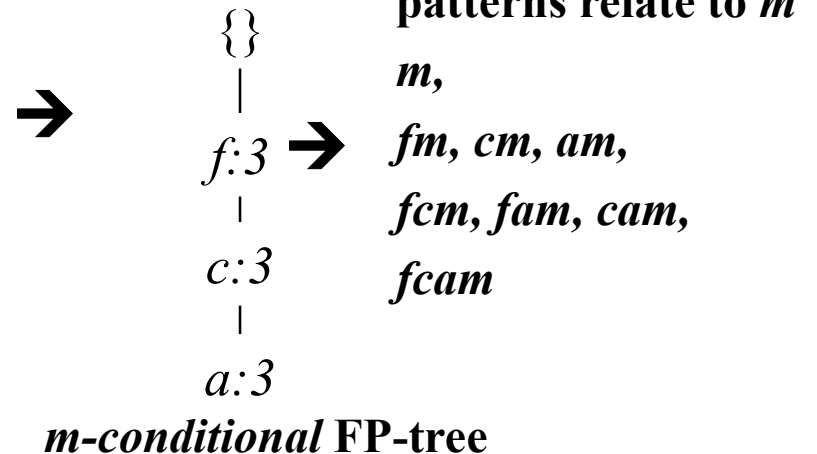
<i>item</i>	<i>cond. pattern base</i>
c	$f:3$
a	$fc:3$
b	$fca:1, f:1, c:1$
m	$fca:2, fcab:1$
p	$fcam:2, cb:1$

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



Mining Frequent Patterns by Creating Conditional Pattern-Bases

Item	Conditional pattern-base	Conditional FP-tree
p	$\{(fca:2), (cb:1)\}$	$\{(c:3)\} p$
m	$\{(fca:2), (fcab:1)\}$	$\{(f:3, c:3, a:3)\} m$
b	$\{(fca:1), (f:1), (c:1)\}$	Empty
a	$\{(fc:3)\}$	$\{(f:3, c:3)\} a$
c	$\{(f:3)\}$	$\{(f:3)\} c$
f	Empty	Empty

An example

- Transaction data

- Assume:

minsup = 30%

minconf = 80%

- An example **frequent itemset**:

{Chicken, Clothes, Milk} [sup = 3/7]

- Association rules** from the itemset:

Clothes → Milk, Chicken [sup = 3/7, conf = 3/3]

...

...

Clothes, Chicken → Milk, [sup = 3/7, conf = 3/3]

t1: Beef, Chicken, Milk

t2: Beef, Cheese

t3: Cheese, Boots

t4: Beef, Chicken, Cheese

t5: Beef, Chicken, Clothes, Cheese, Milk

t6: Chicken, Clothes, Milk

t7: Chicken, Milk, Clothes