ASSOCIATION ANALYSIS

5.1.1 What is association mining?

Association mining aims to extract interesting correlations, frequent patterns, associations or casual structures among sets of items or objects in transaction databases, relational database or other data repositories. Association rules are widely used in various areas such as telecommunication networks, market and risk management, inventory control, cross-marketing, catalog design, loss-leader analysis, clustering, classification, etc.

Examples:

Rule form: "Body⇒head [support, confidence]".

Buys (x,"diapers") ⇒buys(x, "beers") [0.5%, 60%]

major(x, "cs") ^ takes(x, "DB") ⇒grade(x, "A") [1%, 75%]

Association rule: basic concepts:

- Given: (1) database of transaction, (2) each transaction is a list of items (purchased by a customer in visit)
- Find: all rules that correlate the presence of one set of items with that of another set of items.
  - E.g., 98% of people who purchase tires and auto accessories also get automotive services done.
  - E.g., Market Basket Analysis
    This process analyzes customer buying habits by finding associations between the different items that customers place in their "Shopping Baskets". The discovery of such associations can help retailers develop marketing strategies by gaining insight into which items are frequently purchased together by customer.
- **Applications**
  - *⇒maintenance agreement (what the store should do to boost maintenance agreement sales)
  - Home electronics ⇒* (what other products should the store stocks up?)
  - Attached mailing in direct marketing
  - Detecting "ping-pong" ing of patients, faulty "collisions"

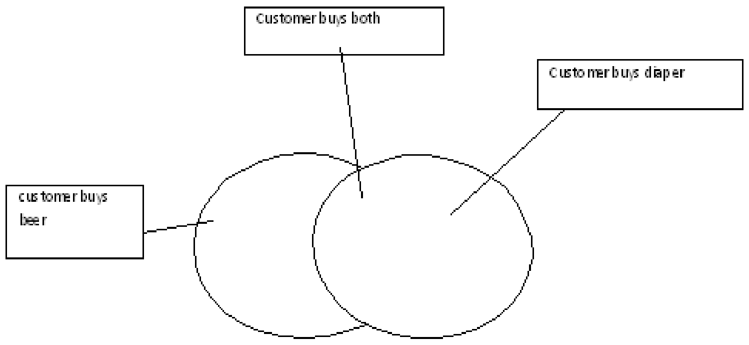
RULE Measures: supports and confidence

Support: percentage of transaction in D that contain AUB.

Confidence: percentage of transaction in D containing A that also contains B.

Support (A ⇒B) = p (A ∪B)
Confidence (A ⇒B) = P (B/A).
Rules that satisfy both a minimum supports threshold (min_sup) and a minimum confidence threshold (min_conf) are called **strong**

Example:



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 ⇒C (50%, 66.6%)

C ⇒A (50%, 100%)

In general, association rules mining can be viewed as a two-step process:

1. Find all frequent itemsets: By definition, each of these itemsets will occur at least as frequently as a predetermined minimum support count, min_sup.
2. Generate strong association rules from the frequent itemsets:
   By definition, these rules must satisfy minimum support and minimum confidence.

Classification of association rules mining:

- **Based on the level of abstraction involved in the rules set:**
  - **Single level association rules** refer items or attribute at only one level. Buys (X, “computer”) ⇒buys(X, “HP printer”)
  - **Multi-level association rules** reference items or attribute at different levels of abstraction.
    Buys(X, “laptop computer”) ⇒ buys(X, “HP printer”)
- **Based on the number of data dimensions involved in the rules:**
  - **Single dimensional Association rule** is an association rule in which items or attribute reference only one dimension.
    Buys (X, “computer”) ⇒ buys (X,”antivirus software”)
  - **Multidimensional association rule** reference two or more dimensions age (X, “30….39”)^income(X, “42k…48k”)⇒buys(X, “high resolution TV”)
- **Based on the types of the values handled in rule:**
  - **Boolean association rule** involve associations between the presence and absence of items.
    buys (X, “SQLServer”) ^ buys (X, “DMBook”) ⇒ buys(X, “DBMiner”)
  - **Quantitative association rule** describe association between quantitative items or attributes.
    Age (X, “30…39”) ^ income(X, “42k…49k”) ⇒ buys(X, “PC”)
- **Based on the kinds of patterns to be mined:**
  - **Frequent itemset mining** is the mining of frequent itemset (sets of items) from transactional or relational data sets.
  - **Sequential pattern mining** searches for frequent subsequence in a sequence data set, where a sequence records an ordering of events.
  - **Structured pattern mining** searches for frequent substructures in a structured data set.
- **Based on various extension to association mining:**
  - Correlation, causality analysis
    Association does not necessarily imply correlation or causality
  - Maxpatterns and closed itemsets
  - Constraints enforced

5.2 Mining single dimensional Boolean association rules from transactional databases:

Approaches for mining association rule are:

- Apriori algorithm
- FP-Growth algorithm

5.2.1 The Apriori algorithm: finding frequent items using candidate generation

Apriori employs an iterative approach known as a level-wise search, where k-item sets are used to explore (k+1)-item sets. First, the set of frequent 1-itemset is found by scanning the database to accumulate the count for each items, and colleting those items that satisfy minimum support. The resulting set is denoted L1. Next, L1 is used to find L2, the set of frequent 2-itemsets, which is used to fined L3, and so on, until no more frequent k-item sets can be found. The finding of each Lk requires one full scan of database.

The Apriori principle: Any subset of a frequent itemset must be frequent The Apriori property follows a two step process: **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.

***Algorithm: Apriori*.** Find frequent itemsets using an iterative level-wise approach based on candidate generation. **Input:** D, a database of transactions Min_sup, the minimum support count threshold.

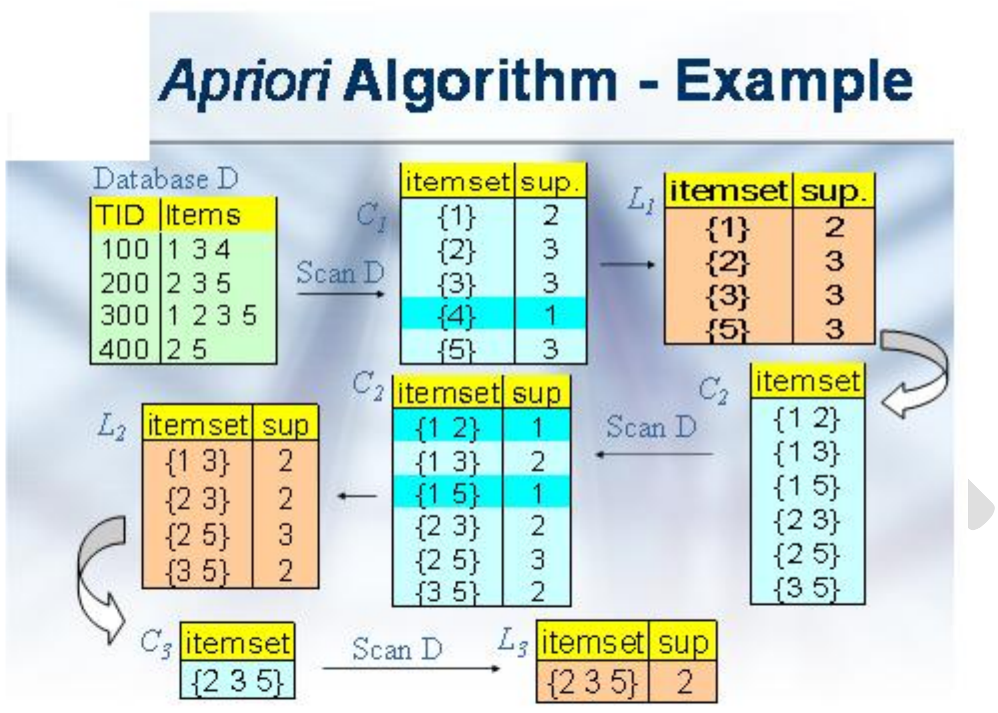
Output: L, frequent itemset in D.
Methods:

```
(1)  L1=find_frequent_1-itemsets(D);
(2)  For (k=2; $L_{k-1} \neq Ø: K++$)
(3)  Ck= Apriori_genfor each transaction (Lk-1);
(4)  for each transaction t€D{// scan D for count
(5)  Ct = subset (Ck,t); // get the subset of t that are candidate
(6)  For each candidate c€Ct,
(7)  c.count++;
(8)  }
(9)  Lk =n { c€Ck|c.count>=min_sup
(10)     }
(11)     Return L = UkLk;
     Procedure aoriori_gen (Lk-1: frequent (k-1)-itemset)
     (1)  for each itemset l1€Lk-1
     (2)  For each itemset l2 € Lk-1
     (3)  If (l1 [1] = l2 [1] ^ (l1 [2] =l2 [2]) ^....^(l1[k-2]=l2[k-2])^(l1[k-1]<l2[k-1]) then {
     (4)  C=l1 ∞ l2; // join step: generate candidates
     (5)  If has_infrequent_subset(c, Lk-1) then
     (6)  Delete c, // prune step: remove unfruitful candidate
     (7)  Else add c to Ck;
     (8)  }
     (9)  Return Ck;
          Procedure has_infrequent_subset (c1 candidate k-itemset;
          Lk-1: frequent (k-1) –itemset; //use prior knowledge
```

(1) For each (k-1) –subset s of c
(2) If s€L_{k-1} then
(3) Return TRUE;
(4) Return FALSE;

THE APRIORI ALGORTHIM FOR DISCOVERING FREQUENT ITEMS FOR MINING BOOLEAN ASSOCIATION RULES.

EXAMPLE 1: There are four transactions in these database, |D|=4 and minimum support count is taken as 2. Use the Apriori algorithm for finding frequent item sets in D.


Apriori Algorithm - Example

Database D

TID	Items
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

C_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{4}	1
{5}	3

L_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{5}	3

C_2

itemset
{1 2}
{1 3}
{1 5}
{2 3}
{2 5}
{3 5}

C_2

itemset	sup
{1 2}	1
{1 3}	2
{1 5}	1
{2 3}	2
{2 5}	3
{3 5}	2

L_2

itemset	sup
{1 3}	2
{2 3}	2
{2 5}	3
{3 5}	2

C_3

itemset
{2 3 5}

L_3

itemset	sup
{2 3 5}	2

Step 1 : Generation and pruning of candidate 1-itemset

Read the database to count the support of 1-itemsets

The set of frequent 1-itemsets L1 is formed after removing the itemset which is less than minimum support. In this example, Item 4 is removed therefore L1 consists of {{1},{2},{3},{5}} Itemset.

Step 2: Generation and pruning of candidate 2-itemset.

a) Join C2=L1∞L1={{1},{2},{3},{5}}*{{1},{2},{3},{5}}

={{1,2},{1,3},{1,5},{2,3},{2,5},{3,5}}.

b) Pruning The 1-item subsets of {1,2} are {1} and {2}, all 1-item subsets of {1, 2} are Members of L1. Therefore, keep {1,2} in C2. Similarly, all nonempty subsets of a frequent itemset is also frequent therefore, C2={{1,2},{1,3},{1,5},{2,3},{2,5},{3,5}} After pruning. Then the set of frequent -2 itemset L2 is formed by counting the support count for C2 and then removing the itemset which is less than minimum threshold.

L2= {{1,3},{2,3},{2,5},{3,5}}

Step 3: Generation and pruning of candidate 3-itemset.

a) Join : C3 =L2∞L2= {{1,3},{2,3},{2,5},{3,5}} * {{1,3},{2,3},{2,5},{3,5}}

={{1,2,3},{1,3,5},{1,2,5},{2,3,5}}

b) Pruning using Apriori property:

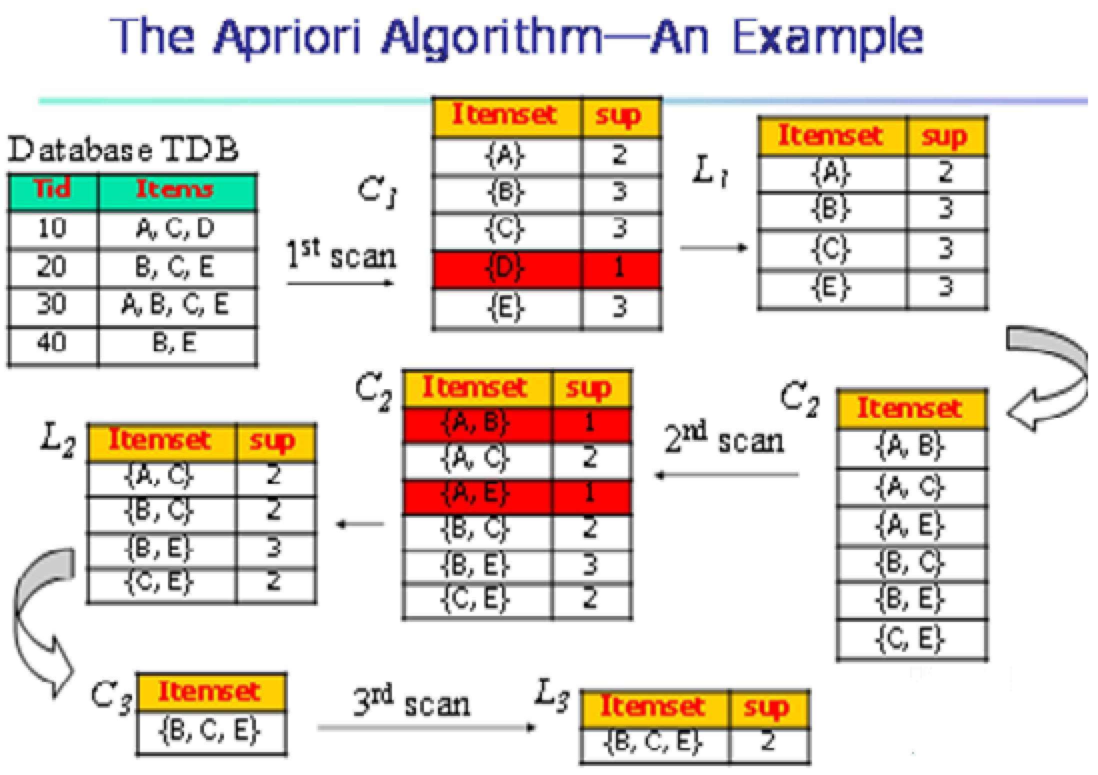
_ The 2-item subsets of {1,2,3} are{1,2},{1,3} and {2,3},{1,2} is not a member Of L2 and so it is not frequent. Therefore remove {1,2,3} from C3.

_ The 2-item subsets of {1,3,5} are {1,3},{1,5} and {3,5},{1,5} is not a member Of L2. Therefore remove {1,3,5} from C3.

_ The 2-item subsets of {1,2,5} are {1,2},{1,5} and {2,5},{1,2} and {1,5} are Not members of L2. Therefore remove {1,2,5} from C3.

_ The 2-item subsets of {2,3,5} are {2,3},{2,5} and {3,5}.All 2-item subsets of {2,3,5} are members of L2. Therefore keep {2,3,5} in C3.
Therefore C3 = {2, 3, 5} after pruning.
Since we have only one itemset in C3, the algorithm terminates, having found all of the frequent item sets.

EXAMPLE 2: There are four transactions in these database, |D|=4 and minimum support count is taken as 2. Use the Apriori algorithm for finding frequent item sets in D.



Database TDB

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

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}

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

C_3

Itemset
{B, C, E}

L_3

Itemset	sup
{B, C, E}	2

5.2.1.1 Generating association rules from frequent item sets:

Once the frequent itemset from transactions in a database D have been found, it is straight forward to generate strong association rules from them (where strong association rules satisfy both minimum support and minimum confidence).

Confidence (A⇒B) = P (B/A) = support_count (A ∪B)/support_count (A)

Association rules based on conditional probability can be generated as follows:

- For each frequent itemset l, generate all nonempty subsets of l.
- For every nonempty subsets of l, output the rule "s ⇒(l-s)" if support_count (l)/support_count(s) ≥ min_conf, where min_conf is the minimum confidence threshold.

5.2.1.2 Methods to improve apriori,s efficiency

- **hash-based itemset counting:** A k-itemset whose corresponding hashing bucket count is below the threshold cannot be frequent
- **transaction reduction:** a transaction that does not contain any frequent k-itemset is useless in subsequent scans
- **partitioning :** any itemset that is potentially frequent in DB must be frequent in at least one of the partition of DB

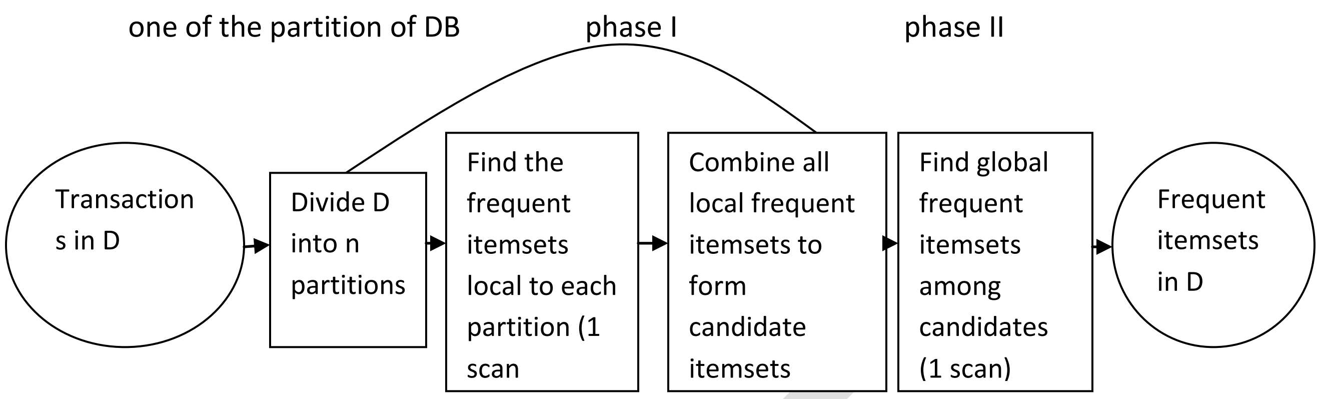

phase I
phase II
Transactions in D
Divide D into n partitions
Find the frequent itemsets local to each partition (1 scan
Combine all local frequent itemsets to form candidate itemsets
Find global frequent itemsets among candidates (1 scan)
Frequent itemsets in D

Fig 5.2 mining by partitioning the data.

- **sampling:** mining on a subset of given data, lower support threshold + a method to determine the completeness
- **dynamic itemset counting:** a new candidate itemsets only when all of their subsets are estimated to be frequent

5.2.1.2 Is Apriori fast enough? _ Performance bottlenecks

- the core of the Apriori algorithm:
  - ✓ Use frequent (k-1)-itemsets to generate candidate k-itemsets
  - ✓ Use database scan and pattern matching to collect counts fro the candidate itemset
- The bottleneck of Apriori: candidate generation
- Huge candidate sets:
  - ✓ 104 frequent 1-itemset will generate 107 candidate 2-itemsets
  - ✓ To discover a frequent pattern of size 100, e.g.,{a_1,a_2,….a_{100}},one needs to generate 2100>>1030 candidates.
- Multiple scans of database:
  - ✓ Needs(n+1) scans, n is the length of the longest patter

5.2.2 Mining frequent patterns without candidate generation

- Compress a large database into a compact, **frequent-pattern tree(FP-tree)** structure
  - ✓ Highly condensed, but complete for frequent pattern mining
  - ✓ Avoid costly database scans
- Develop an efficient, FP-tree-based frequent pattern mining method
  - ✓ A divide-and-conquer methodology: decompose mining tasks into smaller ones
  - ✓ A void candidate generation: sub-database test only.

Benefits of the FP-tree structure

- Completeness
  - ✓ Never breaks a long pattern of any pattern of any transaction
  - ✓ Preserves complete information for frequent pattern mining
- Compactness
  - ✓ Reduce irrelevant information – infrequent items are gone

- ✓ Frequency descending ordering: more frequent items are more likely to be shared
- ✓ Never be larger than the original database (if not count node—links and counts)

Major steps to Mine FP—Tree

1) Construct conditional pattern base for each node in the FP—tree
2) Construct conditional FP—tree from each conditional pattern—base
3) Recursively mine conditional FP—trees an grow frequent patterns obtained so far
)-(If the conditional FP—tree contains a single path, simply enumerate all the patterns

Principals of frequent pattern growth

- Pattern growth property
  - Let a be frequent itemset in DB, B be an‘s conditional pattern base, and b be an itemset in B. Then a E b is a frequent itemset in DB if b is frequent in B.
- “abcdef” is a frequent pattern, if and only if
  - “abcde” is a frequent pattern, and “f” is frequent in the set of transactions containing “abcde”

Algorithm: FP_growth. Mine frequent itemsets using an FP-tree by pattern fragment growth.

Input:

- D, a transaction database.
- Min_sup, the minimum support count threshold.

Output: The complete set of frequent patterns.

method:

1. The FP-Tree is constructed in the following steps:
   (a) Scan the transaction database D once. Collect F, the set of frequent items, and their support counts. Sort F in support count descending order as L, the list of frequent items.
   (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. Let the sorted frequent items list in Trans be [p/P], where p is the first element and P is the remaining list. Call insert_tree ([p/P], T), which is performed as follows. If T has a child N such as that N. item-name=p. item-name=p. item-name then increment N’s count by 1; else create a new node N, and let its count be 1, its parent link be linked to T, and its node-link to the nodes with the same item-name via the node-link structure. If P is nonempty, call insert_tree (P, N) recursively.

2. The Fp-Tree is mined by calling FP-Growth (FP_tree, niull), which is implemented as follows.

```
Procedure FP-growth (Tree, α)
(1) If tree contains a single path P then
(2) For each combination (denoted by β) of the nodes in the path P
(3)  Generate pattern βυα with support_count= minimum support count of nodes in β;
(4) Else for each $a_i$ in the header of tree{
(5) Generate pattern β=$a_i$ U α with support_count = $a_i$. support_count;
(6) Construct β’s conditional pattern base and then β’s conditional FP_tree $Tree_β$;
(7) If $Tree_β$≠ø then
(8) Cal FP_growth ($Tree_β$,β);}
```

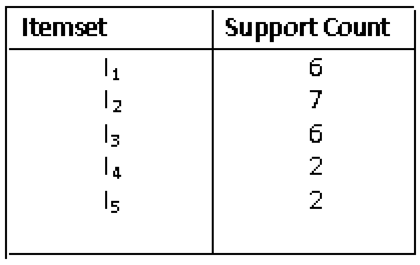
FP Growth

Example2: Finding frequent itemseif without candidate generation (FP Growth)

Steps: (1) FP Tree construction

(2) Creating conditional pattern base.

Input: D→T transaction database, minimum support count, threshold.



Itemset	Support Count
I_1	6
I_2	7
I_3	6
I_4	2
I_5	2

Output: Complete set of frequent pattern.

TID	LIST OF ITEMS_IDS
T_{100}	I_1, I_2, I_5
T_{100}	I_2, I_4
T_{100}	I_2, I_3
T_{100}	I_1, I_2, I_4
T_{100}	I_1, I_3
T_{100}	I_2, I_3
T_{100}	I_1, I_3
T_{100}	I_1, I_2, I_3, I_5
T_{100}	I_1, I_2, I_3

Step1: scan the database D for support count of each candidate
Step2: Sort the items in L-order (descending order)

- The set of frequent items are sorted in the order of descending support.

Itemset	Support Count
I_2	7
I_1	6
I_3	6
I_4	2
I_5	2

Step 3: FP tree construction.

1) Create the root of the as NULL.
2) Scan database D.
3) The item in each transaction proceeded in L-order, i.e;, the order in desending support count and a branch is created for each transaction.For the first transaction T_{100}=I_1, I_2, I_5 which contains 3 items, I_2, I_1, I_5 inL-order. It leads to construction of first branch of tree with 3 node.

In L-order < $I_{2,}$ $I_{1,}$ I_5>,< I_2: 1>,< I_1:1>and < I_5:1>, where 1 support count

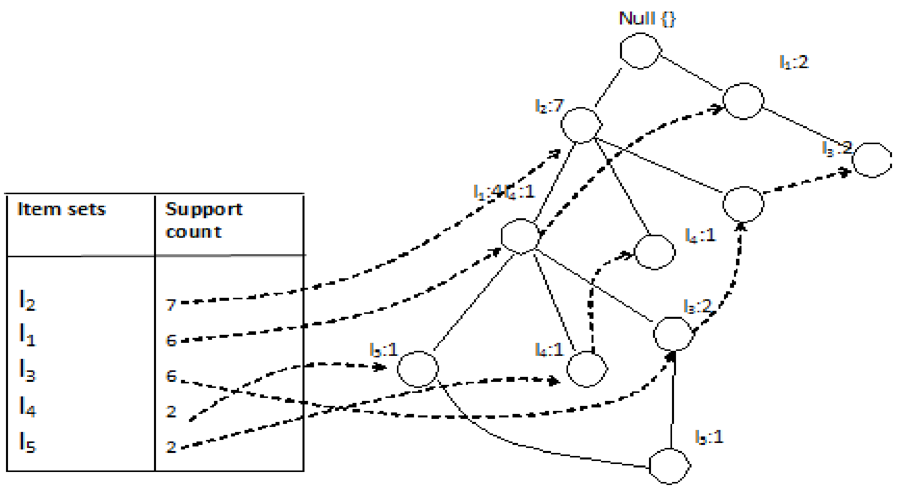


Fig 5.12

Step 4: mining the FP-tree by creating conditional pattern base.

Item	**conditional**	**conditional FP- Tree**	**frequent pattern Generation**
I_5	<I_2, I1.1>	<I_2:2>	<I_2, I_5: 2>
	<I_2, I_1, I3:1>	<I_1:2>	<I_1, I5: 2>
			<I_1, I_2, I5:2>
I_4	<I_2, I1.1>	<I_2:2>	<I_2, I_4: 2>
	<I_2:1>		
I_5	<I_2, I1.2>	<I_2:4, I1:2>	<I_2, I_3: 4><I_1, I3:4>
	<I_2, I_1, I3:2>		
I_1	<I_2:4>	<I_2:4>	<I_2, I1,:4>

The procedure for created a conditional pattern base is:

- Start from the last item in L which has minimum support count.
- Therefore, in the above example, I_5 is the last item in L with minimum support 2.I_5 occurs in two branches namely { $I_{2,}$ $I_{1,}$ I_5 :1} and { $I_{2,}$ $I_{1,}$ $I_{3,}$ I_5 :1}.
- The conditional pattern { $I_{2,}$ I_1 :1} and { $I_{2,}$ $I_{1,}$ I_3 :1} are formed considering I_5 as suffix
- Then the support count for each item in conditional pattern base is calculated to create conditional FP –Tree.
- I_2 and I_1 forms the conditional FP-Tree and I_3 is rejected, since its support count is less than the required minimum support(2)
- Next all frequent pattern corresponding to suffix I_5 are generated buy considering all possible combinations of I_5 and conditional FP-Tree:{ $I_{2,}$ I_5:2}, { $I_{1,}$ I_5:2}, { I_2, I_1, I_5:2}.
- Similarly for I_4 ,IPS two prefix paths from conditional pattern base,{{ I_2, I_1 :2}, { I_2 :1}}, which generates single node conditional FP-Tree, (I_2:2) and derives one frequent pattern ,{ I_2, I_1:2}.

Why is frequent pattern growth fast?

- Our performance study shows
  - FP –growth is an order of magnitude faster than apriori, and is also faster than tree-projection
- Reasoning
  - No candidate generation, no candidate test
  - Use compact data structure
  - Eliminate repeated database scan
  - Basic operation is counting and FP-tree building

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 pS
  Group by p.custID, p.itemID
  Having sum (p.qty)>=10
- Compute iceberg queries efficiently by Apriority:
  - First compute lower dimensions
  - Then compute higher dimensions only when all the lower ones are above the threshold

5.3 MULTIPLE-LEVEL ASSOCIATION RULES

Data mining systems should provide capabilities to mine association rules at multiple levels of abstraction and traverse easily among different abstraction spaces.

- Items often form 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.

Multi-Level Mining: Progressive Deeping

- A top-down, progressive deepening approach:
  - First mine high – level frequent items:
    Milk (15%), bread (10%)
  - Then mine their lower-level "weaker" frequent itemsets:
    2% milk (5%), wheat bread (4%)
- Different min_support threshold across multi-levels lead to different algorithms:
  - If adopting reduced *min_support* at lower levels then examine only those descendents whose ancestor's support is frequent/non – negligible.

Progressive Refinement of Data Mining Quality

- Why progressive refinement?
  - Mining operator can be expensive or cheap, fine or rough
  - Trade speed with quality: step-by-step refinement.
- Superset coverage property:
  - Preserve all the positive answers – allow a positive false test but not a false negative test.

- Two – or multi-step mining
  - First apply rough/cheap operator (superset coverage)
  - Then apply expensive algorithm on a substantially reduced candidate set.

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}

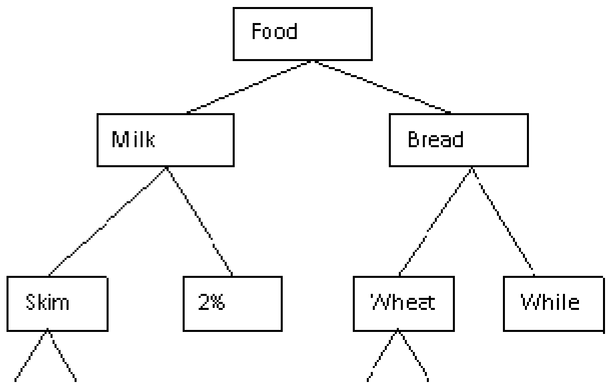


Figure 5.19 multilevel mining5.3.1

5.3.1 Approaches to Mining Multilevel Association Rules:

- **Uniform support:** The same minimum support threshold is used for all levels.
  - Users are required to specify only one minimum support threshold.
  - No need to examine itemset containing any item whose ancestors do not have minimum support.
  - Lower level items do not occur as frequently.
  - If support threshold.
    - Too high => miss low level associations
    - Too low => generate too many level associations

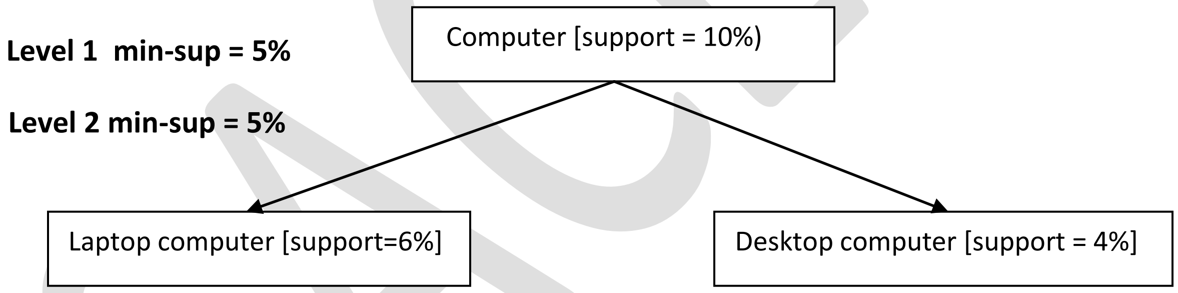


Figure 5.20 Multilevel Mining with Uniform Support

- **Reduced Support:** Each level of abstraction has its own minimum support threshold.

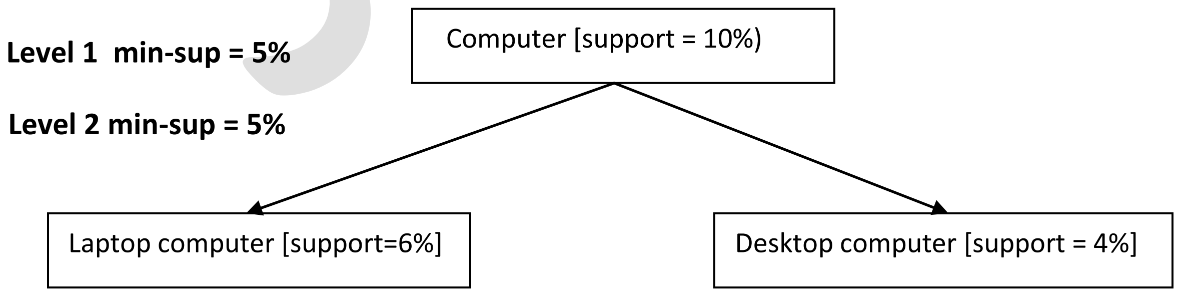


Figure 5.21 Multilevel Mining with reduced Support

Strategies for mining multiple-level association rules using reduced support:

- ***Level-by-level independent –*** Here each node is examined, regardless of whether or not its parent node is found to be frequent.
- ***Levels-cross filtering by k-itemset –*** A *k*-itemset at the ith level is examined if and only if its corresponding parent *k*-itemset at the (i-1) th level is frequent.
- ***Level-cross filtering by single item –*** An item at the *i*th level is examined if and only if its parent node is frequent.

- ***Controlled level-cross filtering by single item -*** In this method, the children of items that do not satisfy the minimum threshold is examined if these items satisfy the level passage threshold.

5.3.2 Multi-level Association: Redundancy Filtering

- Some rules may be redundant due to "ancestor" relationships between items.
- **Example**
  - **Milk => wheat bread [support = 8%, confidence = 70%]**
  - **2% milk => 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.