UNIT-II

ASSOCIATION RULES

2.1 Introduction:

- ARM is to find out association rules or Frequent patterns or subsequences or correlation relationships among large set of data items that satisfy the predefined minimum support and confidence from a given database.

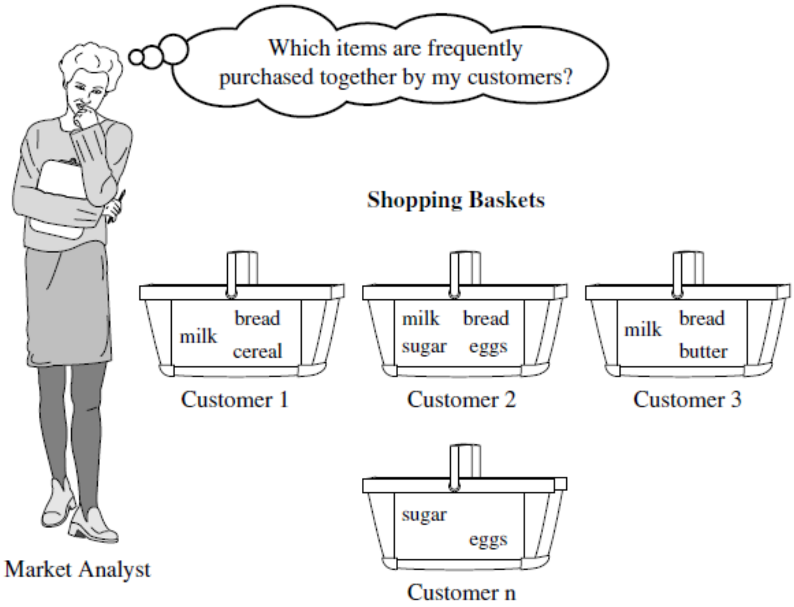
- **Frequent patterns** are patterns (such as itemsets, subsequences, or substructures) that appear in a data set frequently. For example, a set of items, such as milk and bread, that appear frequently together in a transaction data set is a *frequent itemset*. A subsequence, such as buying first a PC, then a digital camera, and then a memory card, if it occurs frequently in a shopping history database, is a (*frequent*) *sequential pattern*. A *substructure* can refer to different structural forms, such as subgraphs, subtrees, or sublattices, which may be combined with itemsets or subsequences.

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.

2.1.1 Market Basket Analysis:

Frequent itemset mining leads to the discovery of associations and correlations among items in large transactional or relational data sets. A typical example of frequent itemset mining is 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 customers.



2.1.2 ***Frequent Itemsets, Closed Itemsets, and Association Rules***

Let I ={*I*1, *I*2, ... , *Im*} be a set of items. Let *D*, the task-relevant data, be a set of database transactions where each transaction *T* is a set of items such that *T* _ I . Each transaction is associated with an identifier, called TID.

An association rule is an implication of the form $A \Rightarrow B$, where $A \subset$ I , $B \subset$ I , and $A \cap B$=f. The rule $A \Rightarrow B$ holds in the transaction set *D* with support *s*, where *s* is the percentage of transactions in *D* that contain $A \subset B$ (i.e., the *union* of sets *A* and *B*, or say, both *A* and *B*). The rule $A \Rightarrow B$ has confidence *c* in the transaction set *D*, where *c* is the percentage of transactions in *D* containing *A* that also contain *B*. This is taken to be the conditional probability, $P(B|A)$. That is,

$$\begin{aligned} support(A \Rightarrow B) &= P(A \cup B) \\ confidence(A \Rightarrow B) &= P(B|A). \end{aligned}$$

If the relative support of an itemset *I* satisfies a prespecified minimum support threshold (i.e., the absolute support of *I* satisfies the corresponding minimum support count threshold), then *I* is a frequent itemset. The set of frequent *k*-itemsets is commonly denoted by *Lk*.

$$confidence(A \Rightarrow B) = P(B|A) = \frac{support(A \cup B)}{support(A)} = \frac{support\_count(A \cup B)}{support\_count(A)}.$$

Support is used to eliminate uninteresting rules. Low support is likely to be uninteresting from a business perspective because it may not be profitable to promote items.

Confidence measures the reliability of the inference made by a rule. For rule A→B, the higher confidence , the more likely it is for **B** to be present in transactions that contain **A**. It is used to estimate conditional probability of **B** given **A.**

In general, association rule 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.

2.2 The Apriori Algorithm:

Apriori is a seminal algorithm proposed by R. Agrawal and R. Srikant in 1994 for mining frequent itemsets for Boolean association rules. Apriori employs an iterative approach known as a *level-wise* search, where *k*-itemsets are usedtoexplore (*k*+1)-itemsets. First, the setof frequent 1-itemsets is found by scanning the database to accumulate the count for each item, and collecting those items that satisfy minimum support. The resulting set is denoted *L*1.Next, *L*1 is used to find *L*2, the set of frequent 2-itemsets, which is used to find *L*3, and so on, until no more frequent *k*-itemsets can be found. The finding of each *Lk* requires one full scan of the database.

Apriori Property:

1. *It makes use of "Upward Closure property" (Any superset of infrequent itemset is also an infrequent set). It follows Bottom-up search, moving upward level-wise in the lattice.*
2. *It makes use of "downward closure property"(any subset of a frequent itemset is a frequent itemset).*
3. *If* Support of an itemset exceeds the support of its subsets, then it is known as the anti-monotone property of support.

2.2.1 Steps in candidate generation:

- ❖ Join Step: To find *Lk*, a set of candidate *k*-itemsets is generated by joining *Lk*-1 with itself.
- ❖ Prune Step: Any k-itemset that is not frequent cannot be a subset of a frequent (k+1)-itemset .

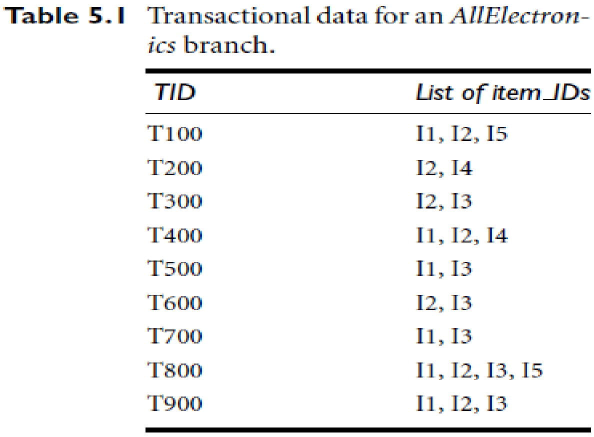


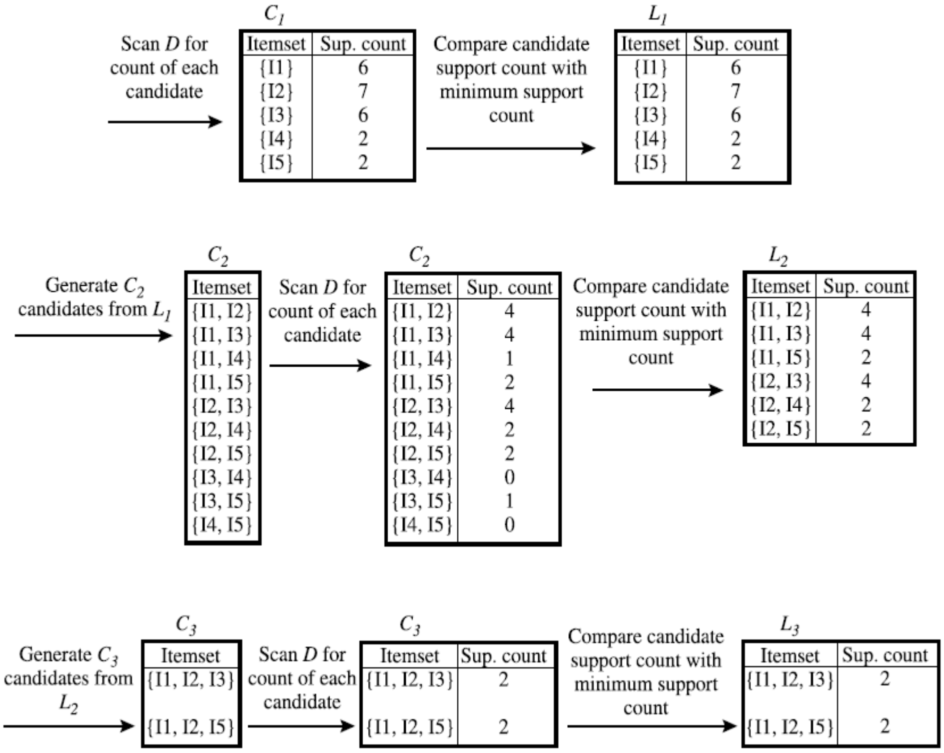
Table 5.1 Transactional data for an *AllElectronics* branch.

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

Based on the *AllElectronics* transaction database,*D*, of Table 5.1

2.2.2 Apriori algorithm steps for finding frequent itemsets in D:-

1. In the first iteration of the algorithm, each item is a member of the set of candidate 1-itemsets, *C*1. The algorithm simply scans all of the transactions in order to count the number of occurrences of each item.
2. Suppose that the minimum support count required is 2, that is, *min sup* = 2. The set of frequent 1-itemsets, *L*1, can then be determined. It consists of the candidate 1-itemsets satisfying minimum support.
3. To discover the set of frequent 2-itemsets, *L*2, the algorithm uses the join *L*1 ⊗ *L*1 to generate a candidate set of 2-itemsets, *C*2.
4. Next, the transactions in*D*are scanned and the support count of each candidate itemset in*C*2 is accumulated, as shown in the middle table of the second row in Figure 5.2.


Scan D for count of each candidate
Compare candidate support count with minimum support count
Generate C_2 candidates from L_1
Scan D for count of each candidate
Compare candidate support count with minimum support count
Generate C_3 candidates from L_2
Scan D for count of each candidate
Compare candidate support count with minimum support count

C_1

Itemset	Sup. count
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2

L_1

Itemset	Sup. count
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2

C_2

Itemset
{I1, I2}
{I1, I3}
{I1, I4}
{I1, I5}
{I2, I3}
{I2, I4}
{I2, I5}
{I3, I4}
{I3, I5}
{I4, I5}

C_2

Itemset	Sup. count
{I1, I2}	4
{I1, I3}	4
{I1, I4}	1
{I1, I5}	2
{I2, I3}	4
{I2, I4}	2
{I2, I5}	2
{I3, I4}	0
{I3, I5}	1
{I4, I5}	0

L_2

Itemset	Sup. count
{I1, I2}	4
{I1, I3}	4
{I1, I5}	2
{I2, I3}	4
{I2, I4}	2
{I2, I5}	2

C_3

Itemset
{I1, I2, I3}
{I1, I2, I5}

C_3

Itemset	Sup. count
{I1, I2, I3}	2
{I1, I2, I5}	2

L_3

Itemset	Sup. count
{I1, I2, I3}	2
{I1, I2, I5}	2

Figure 5.2 Generation of candidate itemsets and frequent itemsets, where the minimum support count is 2.

5. The set of frequent 2-itemsets, *L*2, is then determined, consisting of those candidate 2-itemsets in *C*2 having minimum support.
6. The generation of the set of candidate 3-itemsets,*C*3, is detailed in Figure 5.3.

(a) Join: $C_3 = L_2 \bowtie L_2$ = {{I1, I2}, {I1, I3}, {I1, I5}, {I2, I3}, {I2, I4}, {I2, I5}} $\bowtie$
{{I1, I2}, {I1, I3}, {I1, I5}, {I2, I3}, {I2, I4}, {I2, I5}}
= {{I1, I2, I3}, {I1, I2, I5}, {I1, I3, I5}, {I2, I3, I4}, {I2, I3, I5}, {I2, I4, I5}}.

(b) Prune using the Apriori property: All nonempty subsets of a frequent itemset must also be frequent. Do any of the candidates have a subset that is not frequent?

- The 2-item subsets of {I1, I2, I3} are {I1, I2}, {I1, I3}, and {I2, I3}. All 2-item subsets of {I1, I2, I3} are members of L_2. Therefore, keep {I1, I2, I3} in C_3.
- The 2-item subsets of {I1, I2, I5} are {I1, I2}, {I1, I5}, and {I2, I5}. All 2-item subsets of {I1, I2, I5} are members of L_2. Therefore, keep {I1, I2, I5} in C_3.
- The 2-item subsets of {I1, I3, I5} are {I1, I3}, {I1, I5}, and {I3, I5}. {I3, I5} is not a member of L_2, and so it is not frequent. Therefore, remove {I1, I3, I5} from C_3.
- The 2-item subsets of {I2, I3, I4} are {I2, I3}, {I2, I4}, and {I3, I4}. {I3, I4} is not a member of L_2, and so it is not frequent. Therefore, remove {I2, I3, I4} from C_3.
- The 2-item subsets of {I2, I3, I5} are {I2, I3}, {I2, I5}, and {I3, I5}. {I3, I5} is not a member of L_2, and so it is not frequent. Therefore, remove {I2, I3, I5} from C_3.
- The 2-item subsets of {I2, I4, I5} are {I2, I4}, {I2, I5}, and {I4, I5}. {I4, I5} is not a member of L_2, and so it is not frequent. Therefore, remove {I2, I4, I5} from C_3.

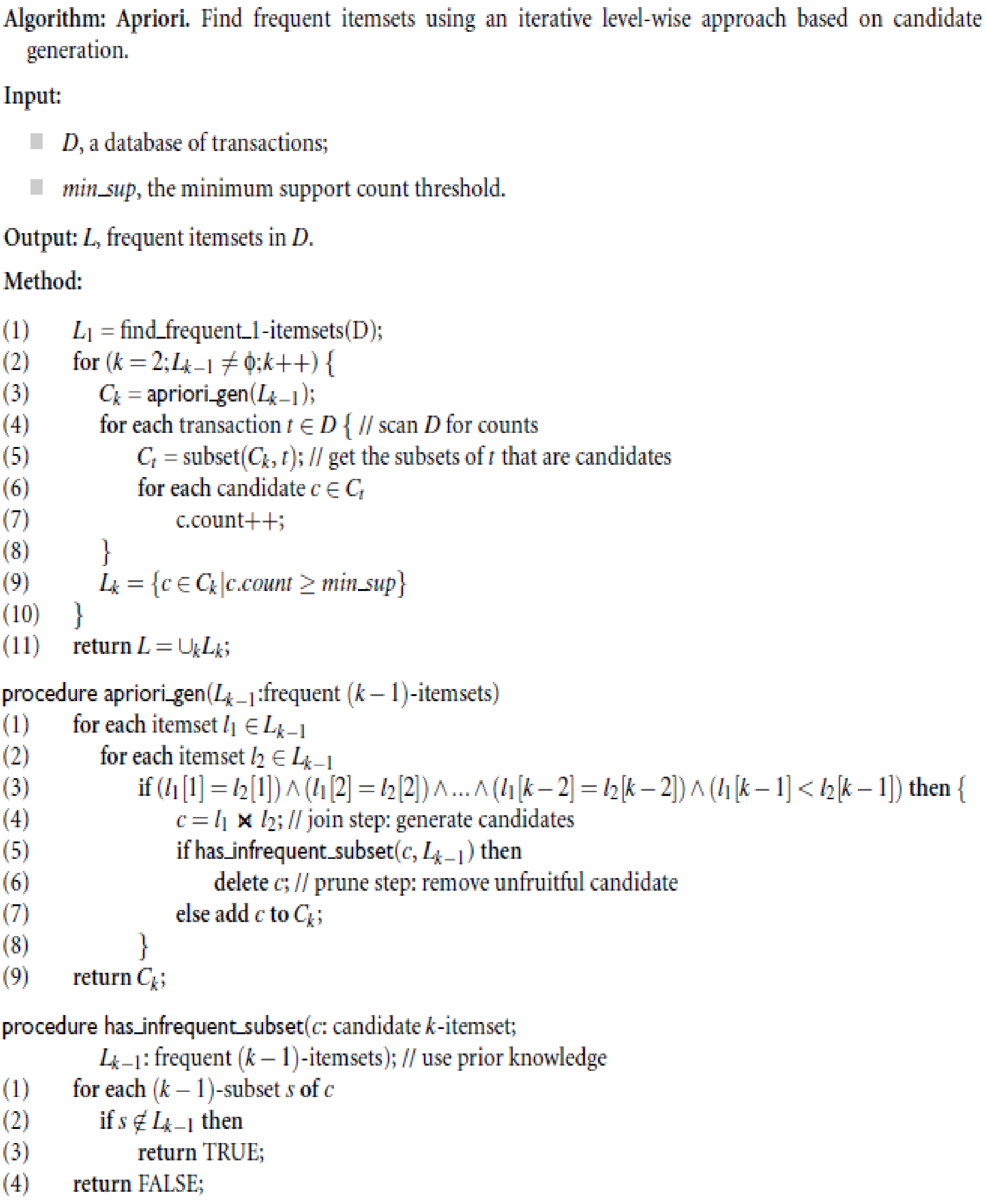
(c) Therefore, C_3 = {{I1, I2, I3}, {I1, I2, I5}} after pruning.

Figure 5.3 Generation and pruning of candidate 3-itemsets, C_3, from L_2 using the Apriori property.

7. The transactions in *D* are scanned in order to determine *L*3, consisting of those candidate 3-itemsets in *C*3 having minimum support (Figure 5.2).

8. The algorithm uses *L*3 ⊗ *L*3 to generate a candidate set of 4-itemsets, *C*4. Although the join results in {I1, I2, I3, I5}, this itemset is pruned because its subset {I2, I3,I5} is not frequent. Thus, *C*4 = f, and the algorithm terminates, having found all of
the frequent itemsets.

2.2.3 pseudo-code for the Apriori algorithm:-



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 itemsets in *D*.

Method:

```
$L_1$ = find_frequent_1-itemsets(D);
for ($k = 2; L_{k-1} \neq \phi; k{+}{+}$) {
    $C_k$ = apriori_gen($L_{k-1}$);
    for each transaction $t \in D$ { // scan $D$ for counts
        $C_t$ = subset($C_k$, $t$); // get the subsets of $t$ that are candidates
        for each candidate $c \in C_t$
            c.count++;
    }
    $L_k = \{c \in C_k | c.count \geq min\_sup\}$
}
return $L = \cup_k L_k$;

procedure apriori_gen($L_{k-1}$:frequent $(k-1)$-itemsets)
for each itemset $l_1 \in L_{k-1}$
    for each itemset $l_2 \in L_{k-1}$
        if $(l_1[1] = l_2[1]) \wedge (l_1[2] = l_2[2]) \wedge ... \wedge (l_1[k-2] = l_2[k-2]) \wedge (l_1[k-1] < l_2[k-1])$ then {
            $c = l_1 \bowtie l_2$; // join step: generate candidates
            if has_infrequent_subset($c, L_{k-1}$) then
                delete $c$; // prune step: remove unfruitful candidate
            else add $c$ to $C_k$;
        }
return $C_k$;

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

2.2.4 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

```
forall itemsets c in $C_k$ do
forall (k-1)-subsets s of c do
if (s is not in $L_{k-1}$) then delete c from $C_k$
```

2.2.5 How to Count Supports of Candidates?

- Why counting supports of candidates a problem?
  - The total number of candidates can be very huge
  - One transaction may contain many candidates
- Method:
  - Candidate itemsets are stored in a *hash-tree*
  - *Leaf* node of hash-tree contains a list of itemsets and counts
  - *Interior* node contains a hash table
  - *Subset function*: finds all the candidates contained in a transaction

Example of Generating Candidates

- 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}*

2.2.6 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 partitions of DB
- Sampling: mining on a subset of given data, lower support threshold + a method to determine the completeness
- Dynamic itemset counting: add new candidate itemsets only when all of their subsets are estimated to be frequent

2.2.7 Is Apriori Fast Enough? — Performance

- The core of the Apriori algorithm:
  - Use frequent (*k* – 1)-itemsets to generate candidate frequent *k*-itemsets
  - Use database scan and pattern matching to collect counts for the candidate itemsets
- The bottleneck of *Apriori*: candidate generation
  - Huge candidate sets:
    - 10^4 frequent 1-itemset will generate 10^7 candidate 2-itemsets
    - To discover a frequent pattern of size 100, e.g., $\{a_1, a_2, \ldots, a_{100}\}$, one needs to generate $2^{100} \approx 10^{30}$ candidates.
  - Multiple scans of database:
    - Needs (*n +1*) scans, *n* is the length of the longest pattern

2.3 FP-Tree Growth Algorithm :(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
  - Avoid candidate generation: sub-database test only!

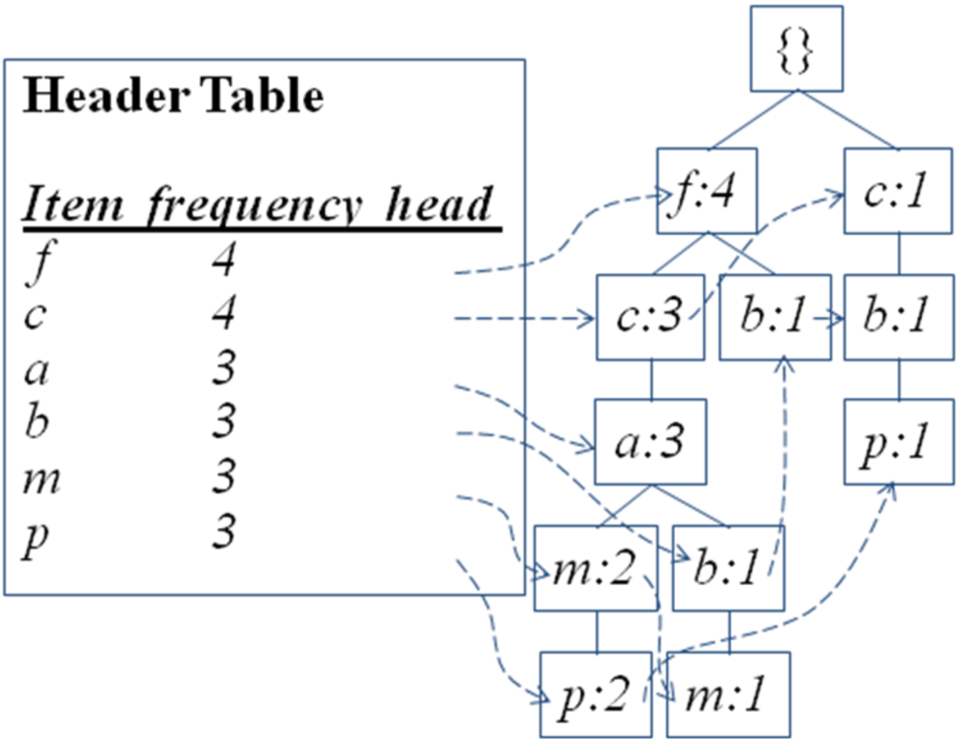
2.3.1 Construct FP-tree from a Transaction DB

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}*	*{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= 0.5

Steps:

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



2.3.2 Benefits of the FP-tree Structure

- Completeness:
  - never breaks a long pattern of any transaction
  - preserves complete information for frequent pattern mining
- Compactness
  - reduce irrelevant information—infrequent items are gone

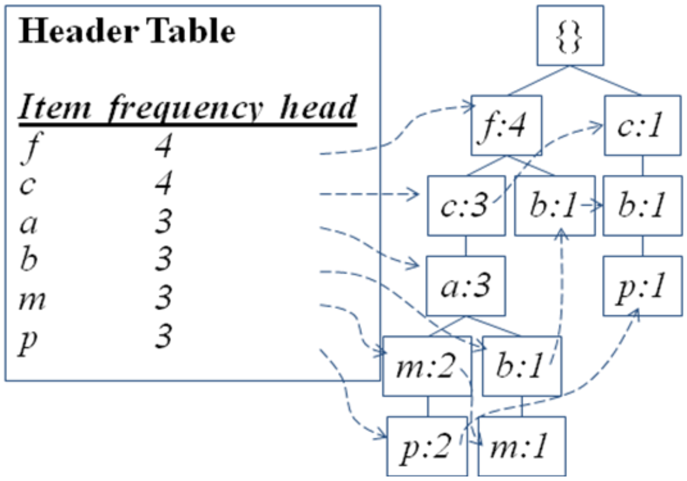
- frequency descending ordering: more frequent items are more likely to be shared
- never be larger than the original database (if not count node-links and counts)
- Example: For Connect-4 DB, compression ratio could be over 100

2.3.3 Mining Frequent Patterns Using FP-tree

- General idea (divide-and-conquer)
  - Recursively grow frequent pattern path using the FP-tree
- Method
  - For each 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)

Step 1: From FP-tree to Conditional Pattern Base

- Starting at the frequent header table in the FP-tree
- Traverse the FP-tree by following the link of each frequent item
- Accumulate all of transformed prefix paths of that item to form a 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*

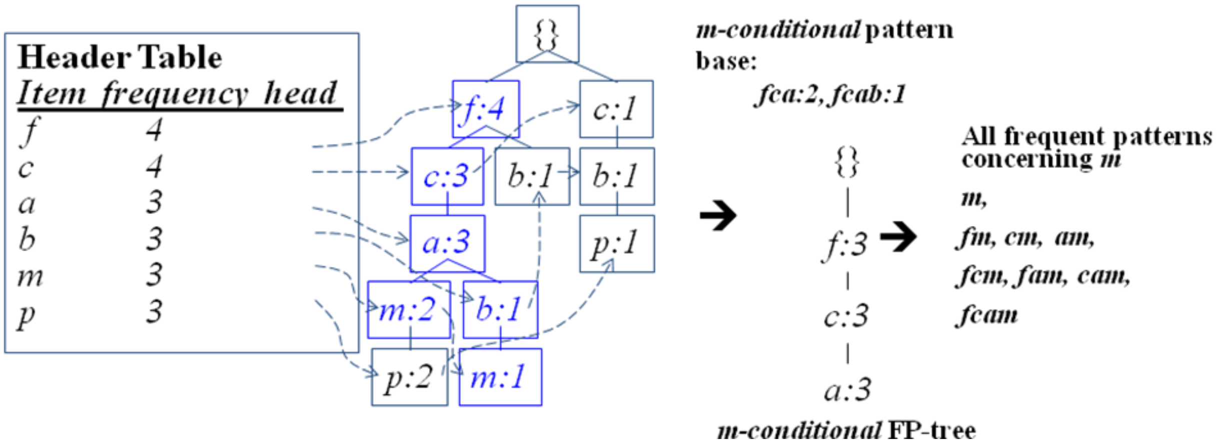
Properties of FP-tree for Conditional Pattern Base Construction

- Node-link property
  - For any frequent item a_i, all the possible frequent patterns that contain a_i can be obtained by following a_i's node-links, starting from a_i's head in the FP-tree header

- Prefix path property
  - To calculate the frequent patterns for a node a_i in a path P, only the prefix sub-path of a_i in P need to be accumulated, and its frequency count should carry the same count as node a_i.

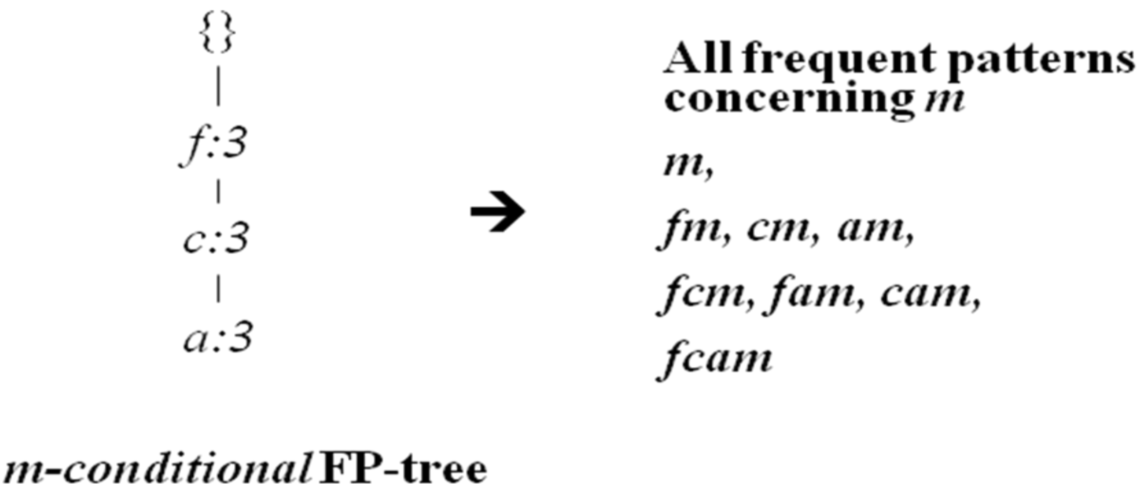
Step 2: Construct Conditional FP-tree

- 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



2.3.4 Single FP-tree Path Generation

- Suppose an FP-tree T has a single path P
- The complete set of frequent pattern of T can be generated by enumeration of all the combinations of the sub-paths of P



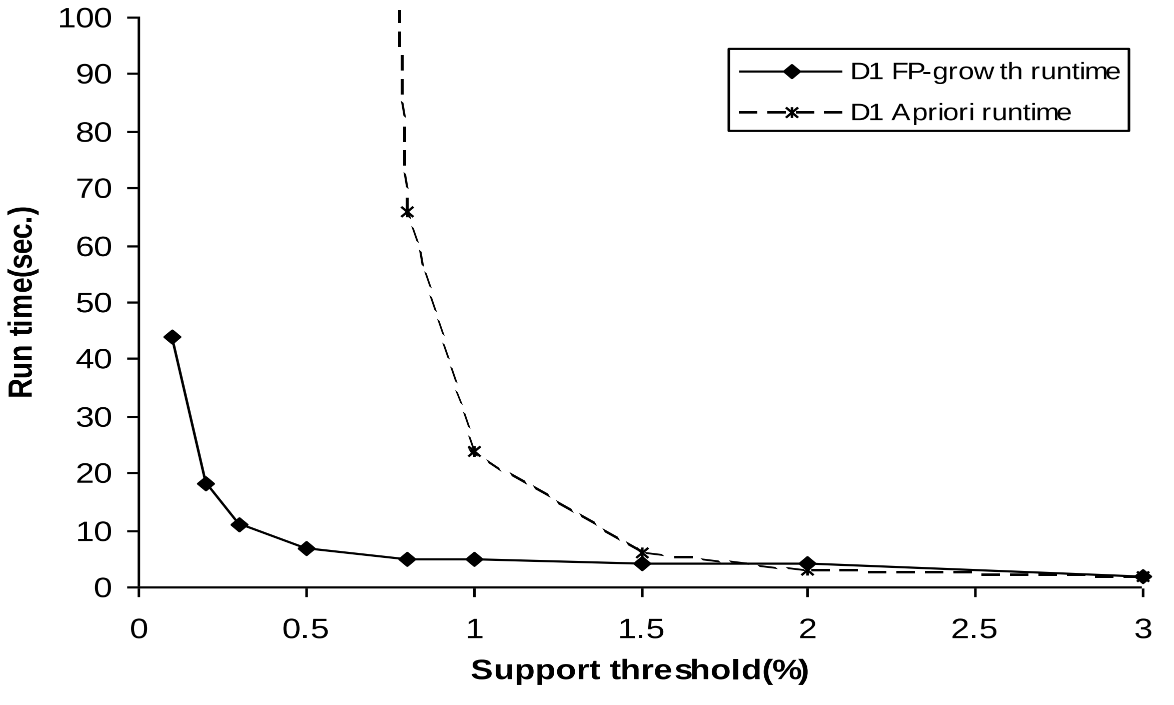
2.3.5 Principles of Frequent Pattern Growth

- Pattern growth property
  - Let α be a frequent itemset in DB, B be α's conditional pattern base, and β be an itemset in B. Then α ∪ β is a frequent itemset in DB iff β 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* "

2.3.6 Why Is Frequent Pattern Growth Fast?

- 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

2.3.7 FP-growth vs. Apriori: Scalability With the Support Threshold



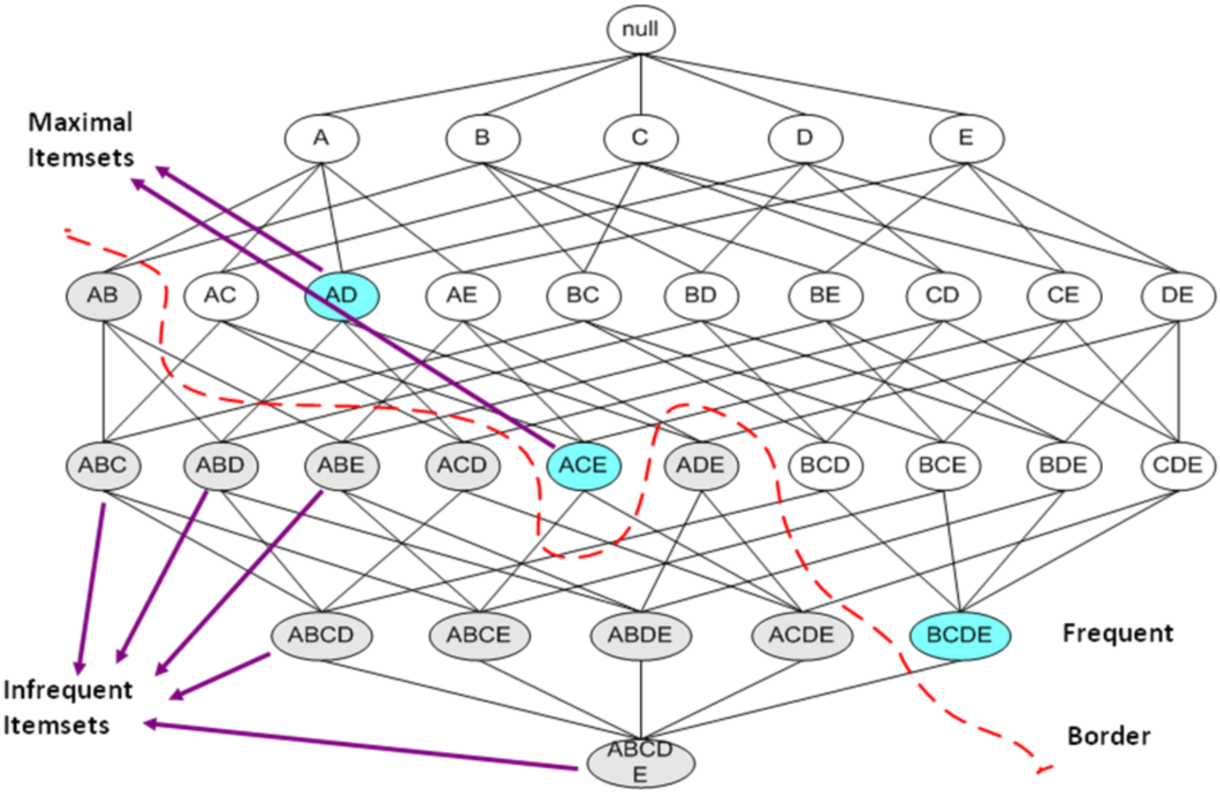
2.4 Compact Representation of Frequent Itemset

- Usually, huge number of frequent itemsets produced from transactional dataset.
- It is useful to identify a **small representative set of itemsets** from which all **other frequent itemsets** can be derived.
- There are two such representations

1. **Maximal Frequent Itemsets:**
   - ➢ An itemset is maximal frequent if none of its immediate supersets is frequent.
2. **Closed Frequent Itemsets:**
   - ➢ An itemset is closed if none of it's immediate supersets has exactly the same support count .
   - ➢ Closed itemsets provide a minimal representation of itemsets without losing their support information.

2.5 Maximal Frequent Itemset

An itemset is maximal frequent if none of its immediate supersets is frequent.

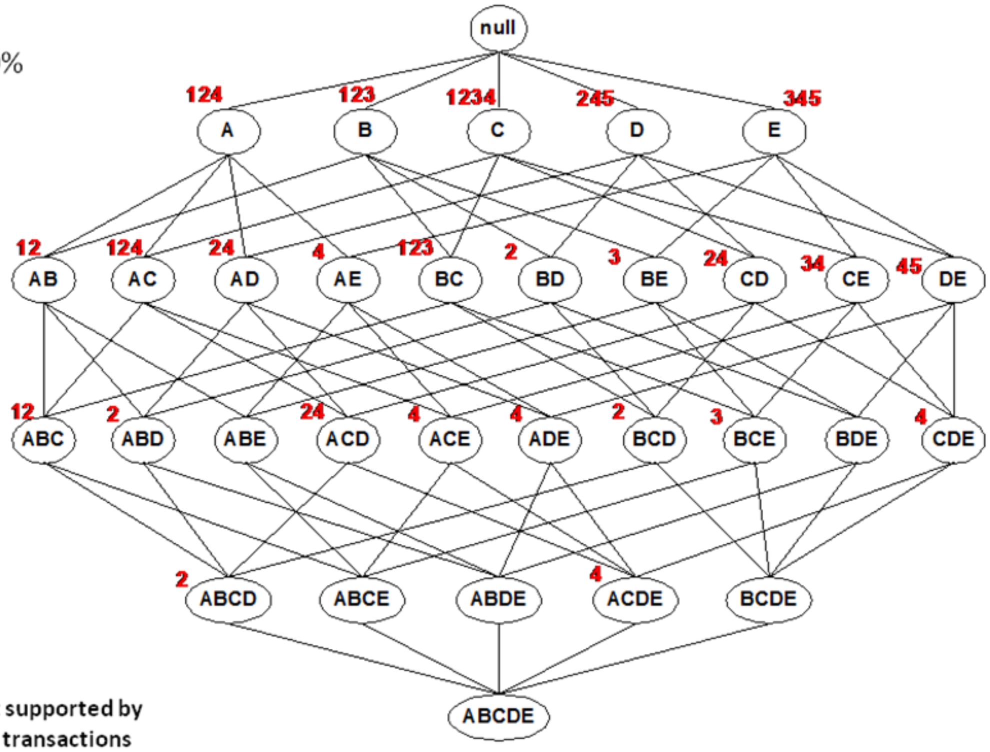


2.6 Closed Frequent Itemsets

Min_support%=40%

TID	Items
1	ABC
2	ABCD
3	BCE
4	ACDE
5	DE

C,D,E,AC,BC,CE,DE,ABC,ACD are the **closed Frequent Itemset**



Not supported by any transactions