Chapter 6: Mining Frequent Patterns, Association and Correlations: Basic Concepts and Methods

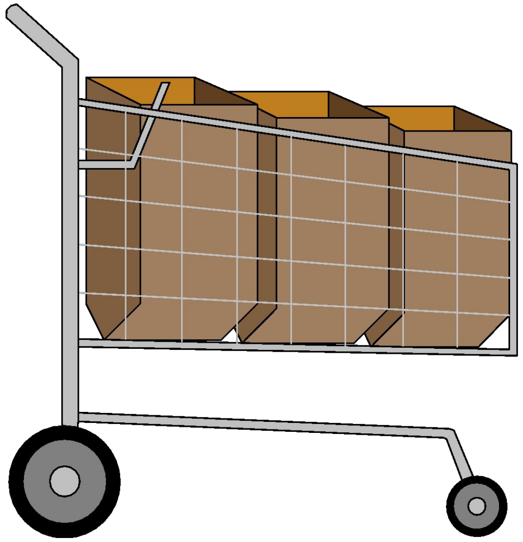
- Basic Concepts
- Frequent Itemset Mining Methods
- Which Patterns Are Interesting?—Pattern Evaluation Methods
- Summary

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 [AIS93] in the context of frequent itemsets and association rule mining
- Motivation: Finding inherent regularities in data
  - What products were often purchased together?— Beer and diapers?!
  - What are the subsequent purchases after buying a PC?
  - What kinds of DNA are sensitive to this new drug?
  - Can we automatically classify web documents?
- Applications
  - Basket data analysis, cross-marketing, catalog design, sale campaign analysis, Web log (click stream) analysis, and DNA sequence analysis.

Market Basket Analysis

- INPUT: list of purchases by purchaser
  - do not have names
- *identify purchase patterns*
  - what items tend to be purchased together
    - obvious: tea-sugar; bread-butter
  - what items are purchased sequentially
    - obvious: phone-cover; computer-mouse
  - what items tend to be purchased by season



Market Basket Analysis

- ***Market Basket Benefits***
  - selection of promotions, merchandising strategy
    - layout or catalogs
    - select products for promotion
    - space allocation, product placement
  - uncover consumer spending patterns
    - correlations: orange juice & waffles
  - joint promotional opportunities

Market Basket Analysis

- Retail outlets
- Telecommunications
- Banks
- Insurance
  - link analysis for fraud
- Medical
  - symptom analysis

Why Is Freq. Pattern Mining Important?

- Freq. pattern: An intrinsic and important property of datasets
- Foundation for many essential data mining tasks
  - **Association**, correlation, and causality analysis
  - **Sequential**, structural (e.g., sub-graph) patterns
  - Pattern analysis in spatiotemporal, multimedia, time-series, and stream data
  - **Classification**: discriminative, frequent pattern analysis
  - **Cluster analysis**: frequent pattern-based clustering
  - **Data warehousing**: iceberg cube and cube-gradient
  - Semantic data compression: fascicles
  - Broad applications

Basic Concepts: Frequent Patterns

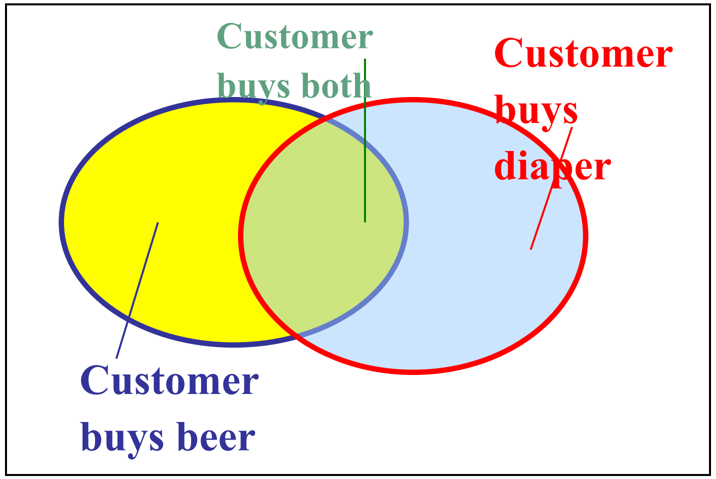
Tid	Items bought
10	Beer, Nuts, Diaper
20	Beer, Coffee, Diaper
30	Beer, Diaper, Eggs
40	Nuts, Eggs, Milk
50	Nuts, Coffee, Diaper, Eggs, Milk

- itemset: A set of one or more items
- k-itemset X = $\{x_1, \ldots, x_k\}$
  - {Beer}, {Diaper}, {Beer,Diaper}, {Beer,Diaper,Egges}

- *(absolute) support*, or, *support count* of X: Frequency or occurrence of an itemset X
  - Beer:3, Nuts:3, Diaper:4, Eggs:3, {Beer, Diaper}:3
- *(relative) support*, *s*, is the fraction of transactions that contains X (i.e., the probability that a transaction contains X)
- An itemset X is *frequent* if X's support is no less than a ***minsup*** threshold

Basic Concepts: Association Rules

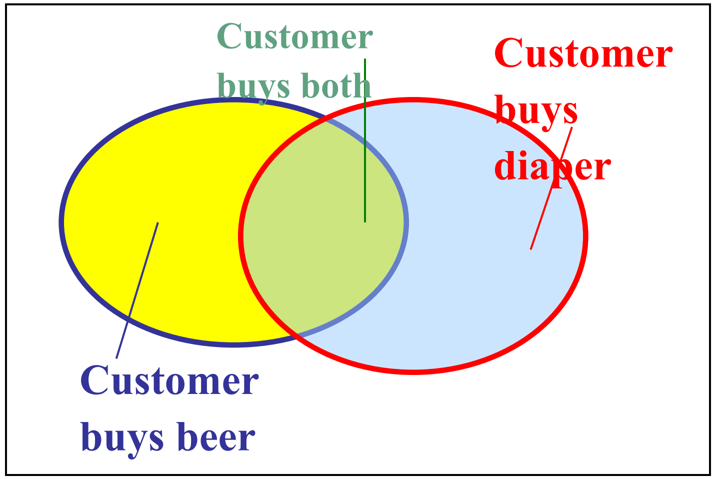
Tid	Items bought
10	Beer, Nuts, Diaper
20	Beer, Coffee, Diaper
30	Beer, Diaper, Eggs
40	Nuts, Eggs, Milk
50	Nuts, Coffee, Diaper, Eggs, Milk



- Find all the rules *X* □ *Y*
  - *{Beer }*□ *{Diaper}*
  - *{Beer,Nuts}* □ *{Milk}*
- Rule Evaluation Matrix
  - support, *s*, probability that a transaction contains X ∪ Y
  - confidence, *c,* conditional probability that a transaction having X also contains *Y*
- Association rules:
  - *Beer → Diaper*
  - $S = \frac{\sigma\,(Beer, Diaper)}{|T|} = \frac{3}{5} = 0.6$
  - $C = \frac{\sigma\,(Beer, Diaper)}{\sigma\,(Beer)} = \frac{3}{3} = 1$

Basic Concepts: Association Rules

Tid	Items bought
10	Beer, Nuts, Diaper
20	Beer, Coffee, Diaper
30	Beer, Diaper, Eggs
40	Nuts, Eggs, Milk
50	Nuts, Coffee, Diaper, Eggs, Milk



- Association rules:
  - *Beer* □ *Diaper* (60%, 100%)
  - *Diaper* □ *Beer* (60%, 75%)
  - {Beer,Diaper} □ *{Eggs}*
- *Let minsup = 50%, minconf = 50%*
  - *Beer* □ *Diaper* (60%, 100%)
  - *Diaper* □ *Beer* (60%, 75%)
- A data set that contains k items can generate upto **2^K-1** item set
- *Total number of possible rules extracted from a data set contains d item is:*

$$R = 3^d - 2^{d+1} + 1.$$

Brute-Force Approach for association rule

- Compute **support** and **confidence** for each rule
- Remove all the rules not satisfying min_support and min_confidence

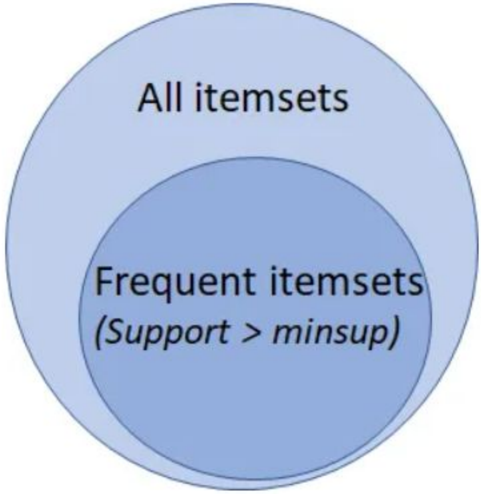
Disadvantage:

- Waste of time in computing rules

Association Rule mining : Two step process

1. Find all frequent itemsets
   - min_support

   E.g.
   - {Bread, Egg, Milk}



2. Generate strong association rules from the frequent itemsets
   - min_support
   - min_confidence

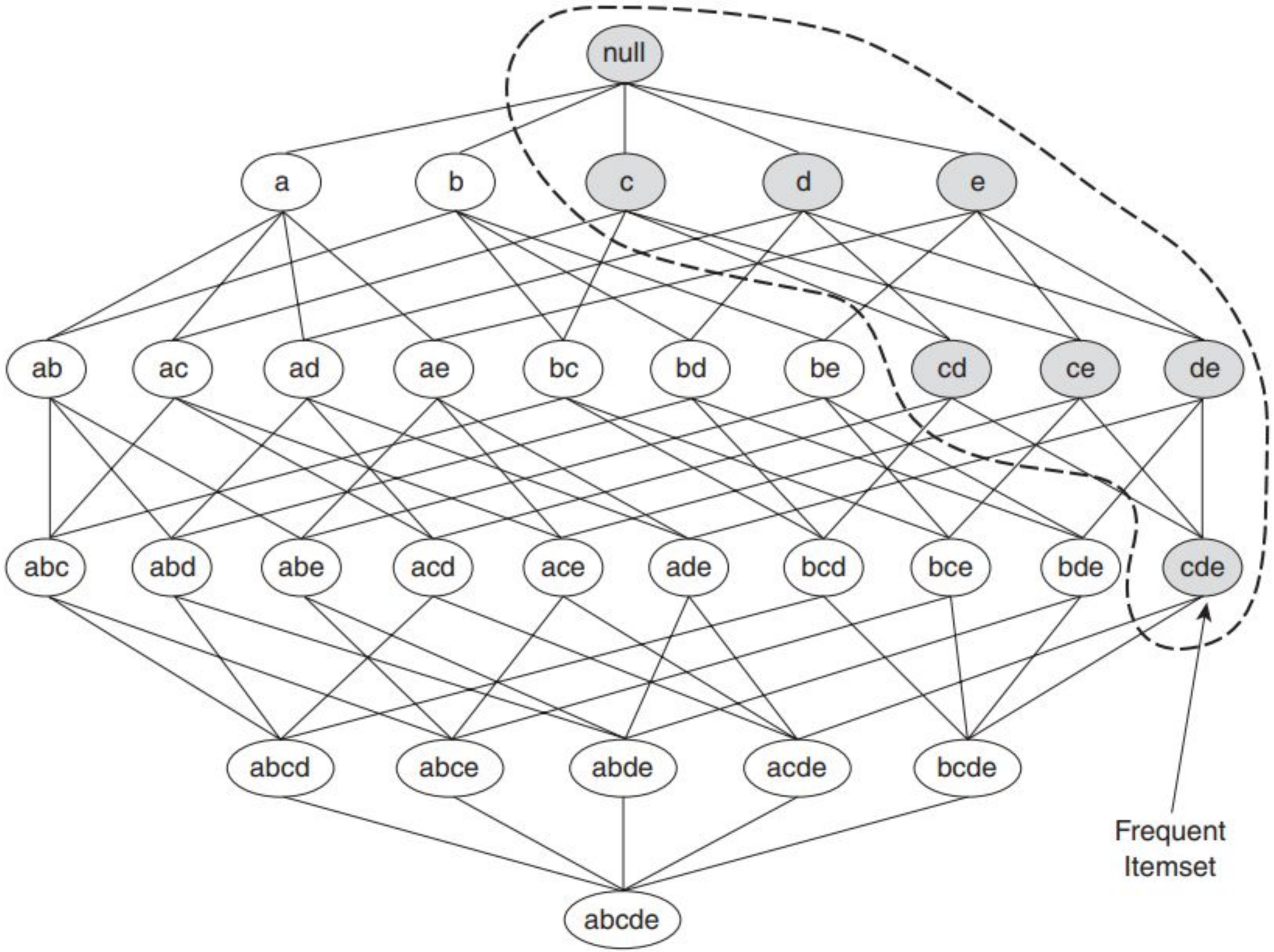
   E.g.
   - {Bread → Egg, Milk},
   - {Bread, Egg → Milk}

1. Frequent Itemset Mining Methods

- 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)

The Downward Closure Property

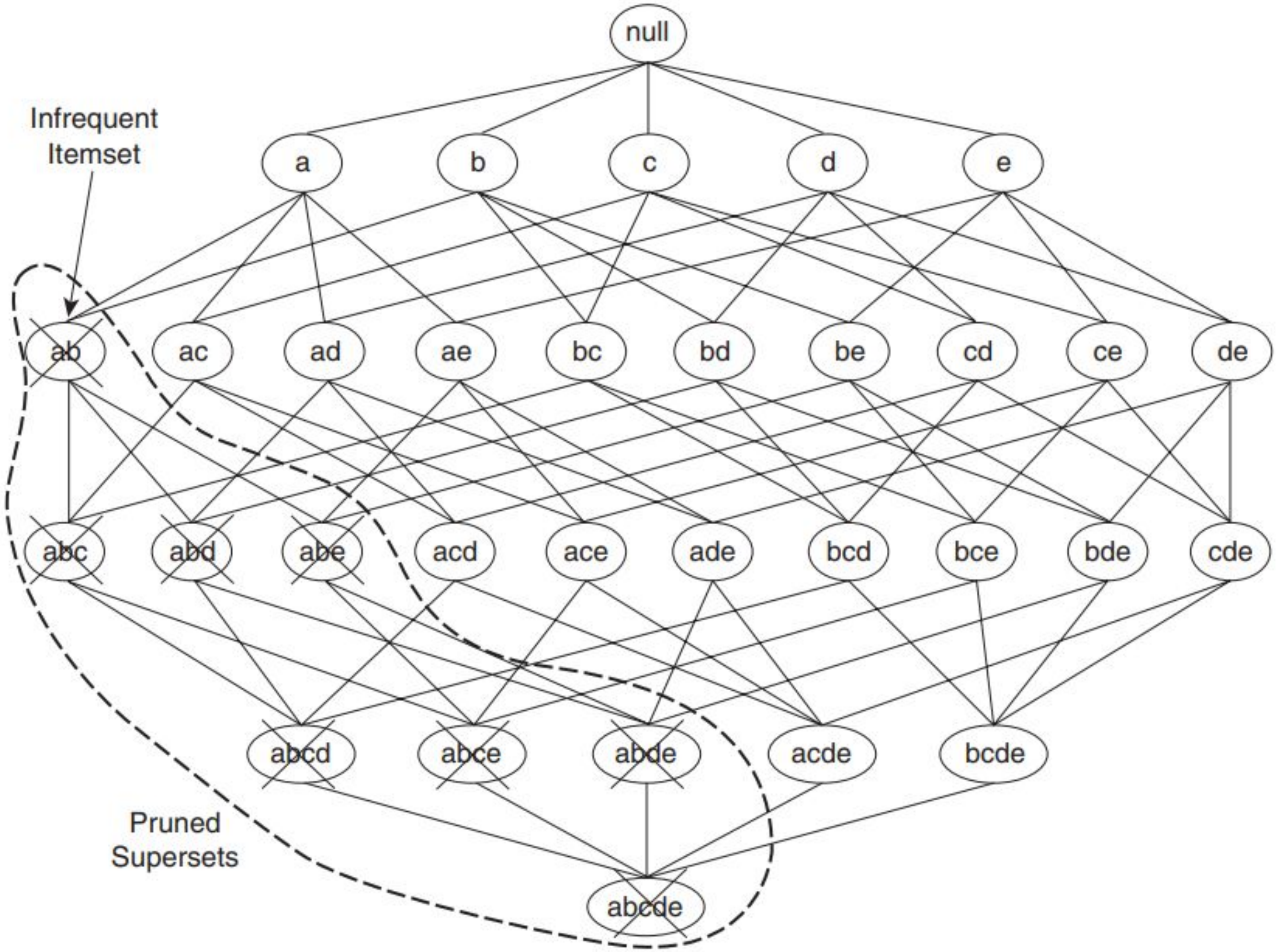
- The downward closure property of frequent patterns
  - Any subset of a frequent itemset must be frequent
  - If **{beer, diaper, nuts}** is frequent, so is **{beer, diaper}**
  - i.e., every transaction having {beer, diaper, nuts} also contains {beer, diaper}



Apriori: A Candidate Generation & Test Approach

- Apriori pruning principle: If there is any itemset which is infrequent, its superset should not be generated/tested!

- Also known as anti-monotone property.
  - If a set cannot pass a test, all of its supersets will fail the same test as well.



Apriori Algorithm

- Used for mining frequent itemsets for Boolean association rules.
- Use priori knowledge (use L1 to design C2)
- Level-wise search (use K-itemsets to explore (K+1)-itemsets)
- 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

$\xrightarrow{1^{st}\text{ scan}}$ C_1

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

$\xrightarrow{L_1}$

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}

$\xleftarrow{2^{nd}\text{ scan}}$ C_2

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

$\leftarrow$ L_2

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

C_3

Itemset
{B, C, E}

$\xrightarrow{3^{rd}\text{ scan}}$ L_3

Itemset	sup
{B, C, E}	2

The Apriori Algorithm (Pseudo-Code) - 1

C_k: Candidate itemset of size k
L_k : frequent itemset of size k

```
L1 = find_frequent_1-itemsets(D);
for (k = 2; Lk−1 ≠ φ; k++) {
    Ck = apriori_gen(Lk−1);
    for each transaction t ∈ D { // scan D for counts
        Ct = subset(Ck, t); // get the subsets of t that are candidates
        for each candidate c ∈ Ct
            c.count++;
    }
    Lk = {c ∈ Ck | c.count ≥ min_sup}
}
return L = ∪k Lk;
```

The Apriori Algorithm (Pseudo-Code) -2

How to generate candidates?

Step 1: self-joining L_k

Step 2: pruning

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;

Implementation of Apriori

- How to generate candidates?
  - Step 1: self-joining L_k
  - Step 2: pruning
- 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*}

Exercise

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

minimum support count is 2

Solution

Scan D for count of each candidate →

C_1

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

Compare candidate support count with minimum support count →

L_1

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

Generate C_2 candidates from L_1 →

C_2

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

Scan D for count of each candidate →

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

Compare candidate support count with minimum support count →

L_2

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

Generate C_3 candidates from L_2 →

C_3

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

Scan D for count of each candidate →

C_3

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

Compare candidate support count with minimum support count →

L_3

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

Generating Association Rule

- The data contain frequent itemset X = {I1, I2, I5}. What are association rules that can be generated from X?
- The non-empty subsets of X are
  - {I1, I2}, {I1, I5}, {I2, I5}, {I1}, {I2}, {I5}
  - {I1, I2} => I5, confidence = 2/4 = 50%
  - {I1, I5} => I2, confidence = 2/2 = 100%
  - {I2, I5} => I1, confidence = 2/2 = 100%
  - I1 => {I2, I5}, confidence = 2/6 = 33%
  - I2 => {I1, I5}, confidence = 2/7 = 29%
  - I5 => {I1, I2}, confidence = 2/2 = 100%

Advantage of apriori over brute_force approach

TID	Items
1	{Bread, Milk}
2	{Bread, Diapers, Beer, Eggs}
3	{Milk, Diapers, Beer, Cola}
4	{Bread, Milk, Diapers, Beer}
5	{Bread, Milk, Diapers, Cola}

Minimum support count = 3

Candidate 1-Itemsets

Item	Count
Beer	3
Bread	4
Cola	2
Diapers	4
Milk	4
Eggs	1

Candidate 2-Itemsets

Itemset	Count
{Beer, Bread}	2
{Beer, Diapers}	3
{Beer, Milk}	2
{Bread, Diapers}	3
{Bread, Milk}	3
{Diapers, Milk}	3

Itemsets removed because of low support

Candidate 3-Itemsets

Itemset	Count
{Bread, Diapers, Milk}	3

itemset generated using brute_force method:

$$\binom{6}{1} + \binom{6}{2} + \binom{6}{3} = 6 + 15 + 20 = 41$$

itemset generated using apriori:

$$6 + 6 + 1 = 13$$

Generating Association Rules from Frequent Itemsets

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

- For each frequent itemset *l*, generate all nonempty subsets of *l*.
- For every nonempty subset *s* of *l*, output the rule "s⇒(l-s)"
  - If $\frac{support\_count(l)}{support\_count(l-s)} \geq \min\_conf$

Improving the Efficiency of Apriori

Limitations:

- Require many database scan
- Assume transaction database is memory resident

Strategies to improve efficiency of Apriori

- Hash-based technique
- Transaction reduction
- Partitioning
- Sampling
- Dynamic itemset counting

Hash-based technique

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

Scan D for count of each candidate →

C_1

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

Compare candidate support count with minimum support count →

L_1

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

Create hash table H_2 using hash function $h(x, y) = ((\mathit{order\ of\ } x) \times 10 + (\mathit{order\ of\ } y)) \bmod 7$ →

H_2

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, H_2, for candidate 2-itemsets. This hash table was generated by scanning Table 6.1's transactions while determining L_1. 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 C_2.

Hash-based technique

Items	Support Count	Order of x
I1	6	1
I2	7	2
I3	6	3
I4	2	4
I5	2	5

ItemSet	Count	HashFunction
I1, I2	4	(1*10+2) mod 7 = 5
I1, I3	4	(1*10+3) mod 7 = 6
I1, I4	1	(1*10+4) mod 7 = 0
I1, I5	2	(1*10+5) mod 7 = 1
I2, I3	4	(2*10+3) mod 7 = 2
I2, I4	2	(2*10+4) mod 7 = 3
I2, I5	2	..
I3, I4	0	..
I3, I5	1	..
I4, I5	0	..

Transaction reduction

- Reducing the number of transactions scanned in future iterations:
- A transaction that does not contain any frequent k-itemsets cannot contain any frequent $(k+1)$-itemsets.
- Therefore, such a transaction can be marked or removed from further consideration because subsequent scans of the database for j-itemsets, where $j > k$, will not require it.

TID	**Items**
T100	I1, I2, I5
T200	I2, I4
T300	I6,I7
T400	I1, I2, I4
T500	I1, I3

Transaction reduction

TID	Items
T1	I1, I2, I5
T2	I2, I3, I4
T3	I3,I4
T4	I1, I2, I3, I4

	I1	I2	I3	I4	I5
T1	1	1	0	0	1
T2	0	1	1	1	0
T3	0	0	1	1	0
T4	1	1	1	1	0

2 3 3 3 0

	I1	I2	I3	I4
T1	1	1	0	0
T2	0	1	1	1
T3	0	0	1	1
T4	1	1	1	1

Min_support = 2, Remove I5

Transaction reduction

TID	Items
T1	I1, I2, I5
T2	I2, I3, I4
T3	I3,I4
T4	I1, I2, I3, I4

	I1,I2	I1,I3	I1,I4	I2,I3	I2,I4	I3,I4
~~T1~~	~~1~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~
T2	0	~~0~~	~~0~~	1	1	1
~~T3~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~1~~
T4	1	~~1~~	~~1~~	1	1	1

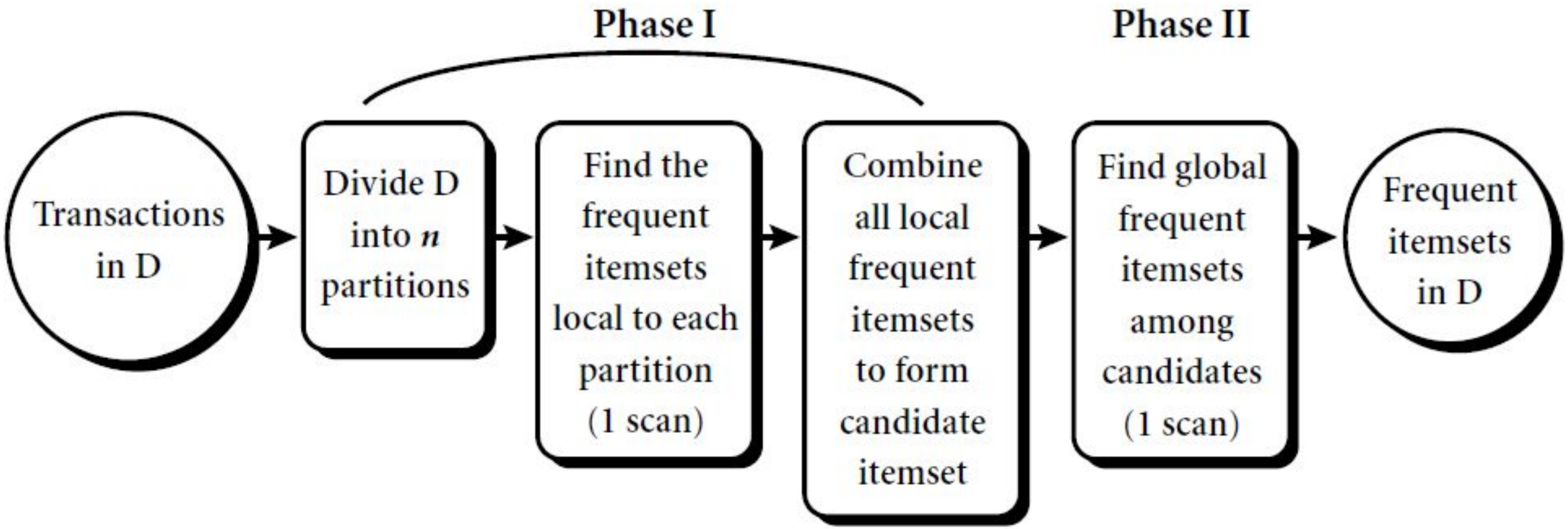
Min_support = 2

	I1,I2	I2,I3	I2,I4	I3,I4
T2	~~0~~	1	1	1
T4	~~1~~	1	1	1

	I1, I2,I3	I1,I2,I4	I2,I3,I4
~~T2~~	~~0~~	~~0~~	~~1~~
T4	~~1~~	~~1~~	1

Partitioning

- Partitioning the data to find candidate itemsets
- It consists of two phases.



Partitioning

- **Phase I:**
  - subdivides the transactions of *D* into *n* no overlapping partitions
  - For each partition, all frequent itemsets within the partition are found.
  - local frequent itemset may or may not be frequent with respect to the entire database, *D*.
  - *Any itemset that is potentially frequent with respect to D must occur as a frequent itemset in at least one of the partitions*.
  - All local frequent itemsets are candidate itemsets with respect to *D*.
  - The collection of frequent itemsets from all partitions forms the global candidate itemsets with respect to *D*.

Partitioning

- **Phase II**
  - second scan of *D* is conducted in which the actual support of each candidate is assessed in order to determine the global frequent itemsets.

- **Advantage:**
  - Partition size and the number of partitions are set so that each partition can fit into main memory and therefore be read only once in each phase.

Sampling for Frequent Patterns

- Select a sample of original database, mine frequent patterns within sample using Apriori
- The sample size of *S* is such that the search for frequent itemsets in *S* can be done in main memory
- L^S is Frequent itemsets local to *S.*
- it is possible that we will miss some of the global frequent itemsets
- Check entire database for L^S ,do we get all the items as a frequent?
- If L^S actually contains all of the frequent itemsets in *D*, then only one scan of *D* is required. Otherwise, a second pass can be done in order to find the frequent itemsets that were missed in the first pass

Dynamic itemset counting (DIC)

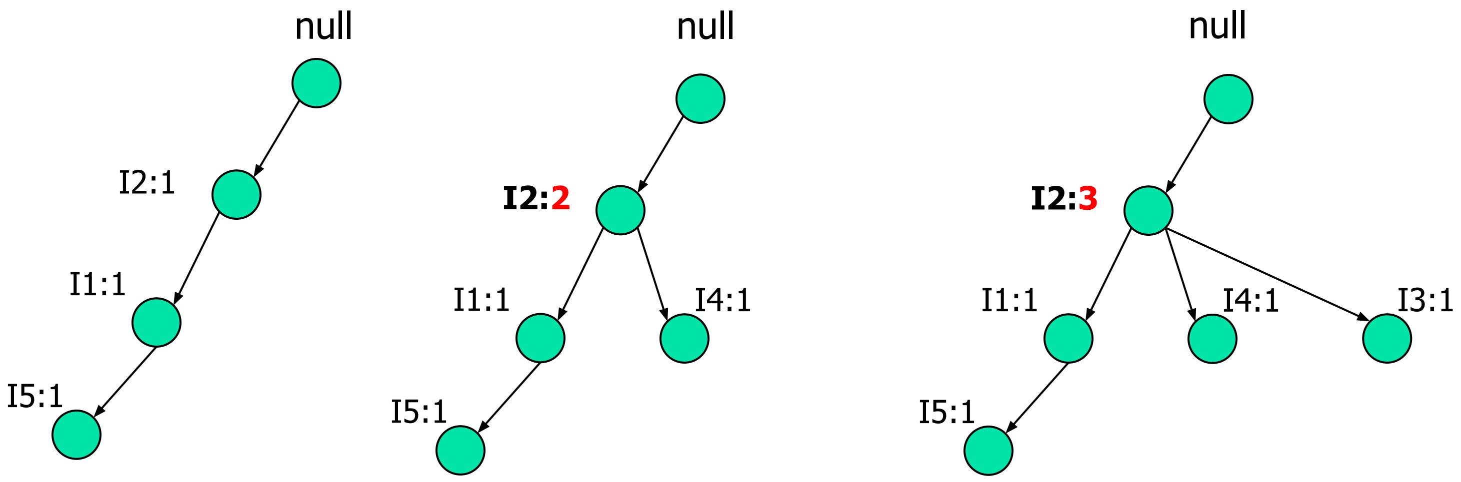
- Apriori algorithm:
  - To identify candidate 3-itemset groupings, it will require a minimum of 3 scans of the database and to identify candidate 4-itemset groupings, it will require four scans of the database
- **Dynamic Itemset Counting (DIC)** algorithm is that it requires only 2 scans of the database to identify frequent item pairs in the whole database.
- On the assumption that data is homogenous throughout in each partition, data is processed for the DIC algorithm.

Pattern-Growth Approach: Mining Frequent Patterns Without Candidate Generation

- Bottlenecks of the Apriori approach
  - Breadth-first (i.e., level-wise) search
  - Candidate generation and test
    - Often generates a huge number of candidates
- The FPGrowth Approach (J. Han, J. Pei, and Y. Yin, SIGMOD' 00)
  - Depth-first search
  - Avoid explicit candidate generation
- Major philosophy: Grow long patterns from short ones using local frequent items only
  - "abc" is a frequent pattern
  - Get all transactions having "abc", i.e., project DB on abc: DB|abc
  - "d" is a local frequent item in DB|abc □ abcd is a frequent pattern

Construct FP-tree from a Transaction Database

1. Scan DB once, find frequent 1-itemset (single item pattern)
2. Sort frequent items in frequency in the order of descending support count. list is denoted L.
3. Scan DB again, construct FP-tree



TID	List of item_IDs	Items After Rearrangement
T100	I1, I2, I5	I2, I1, I5
T200	I2, I4	I2,I4
T300	I2, I3	I2, I3
T400	I1, I2, I4	I2, I1, I4
T500	I1, I3	I1, I3
T600	I2, I3	I2, I3
T700	I1, I3	I1, I3
T800	I1, I2, I3, I5	I2, I1, I3, I5
T900	I1, I2, I3	I2, I1, I3

minimum support = 2

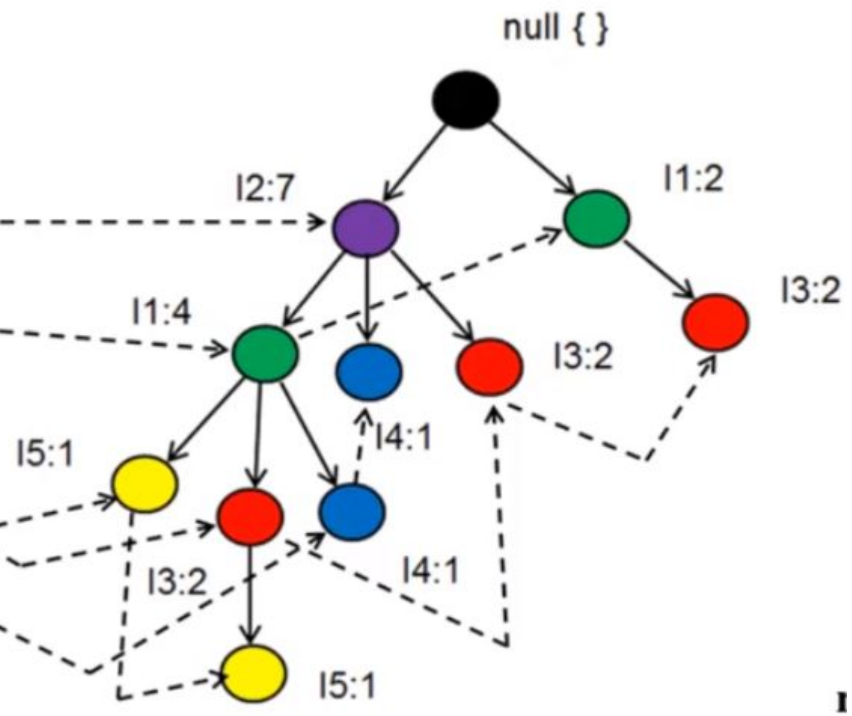
Construct FP-tree from a Transaction Database

1. Scan DB once, find frequent 1-itemset (single item pattern)
2. Sort frequent items in frequency in the order of descending support count. list is denoted L.
3. Scan DB again, construct FP-tree

TID	List of item_IDs	Items After Rearrangement
T100	I1, I2, I5	I2, I1, I5
T200	I2, I4	I2,I4
T300	I2, I3	I2, I3
T400	I1, I2, I4	I2, I1, I4
T500	I1, I3	I1, I3
T600	I2, I3	I2, I3
T700	I1, I3	I1, I3
T800	I1, I2, I3, I5	I2, I1, I3, I5
T900	I1, I2, I3	I2, I1, I3

minimum support = 2

Item Id	Support Count	Node Link
{I2}	7	
{I1}	6	
{I3}	6	
{I4}	2	
{I5}	2	

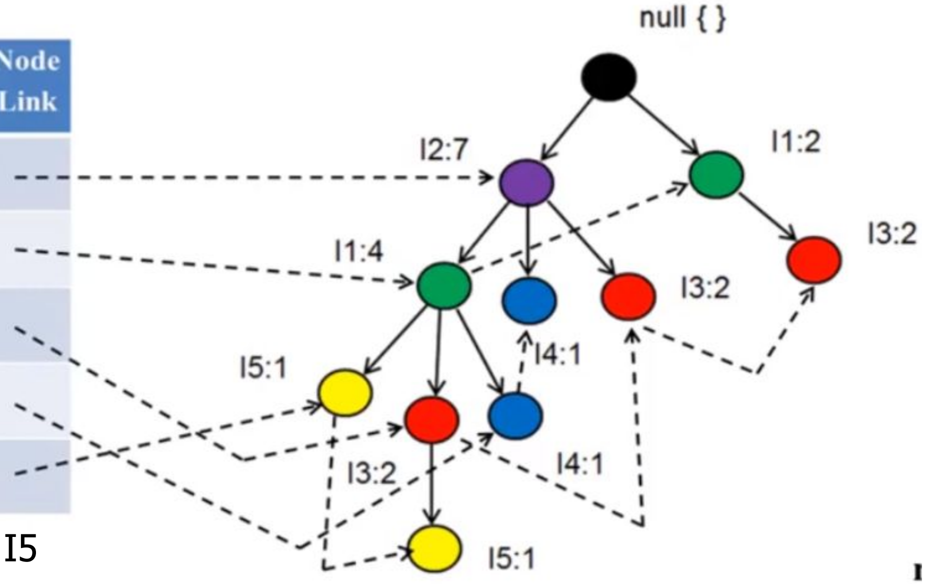


FP-tree Mining

- Start from each frequent length-1 pattern (as an **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),
- construct its **(*conditional*) FP-tree,** and perform mining recursively on such a tree.
- The **pattern growth** is achieved by the concatenation of the suffix pattern with the frequent patterns generated from a conditional FP-tree.

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

Item Id	Support Count	Node Link
{I2}	7	
{I1}	6	
{I3}	6	
{I4}	2	
{I5}	2	

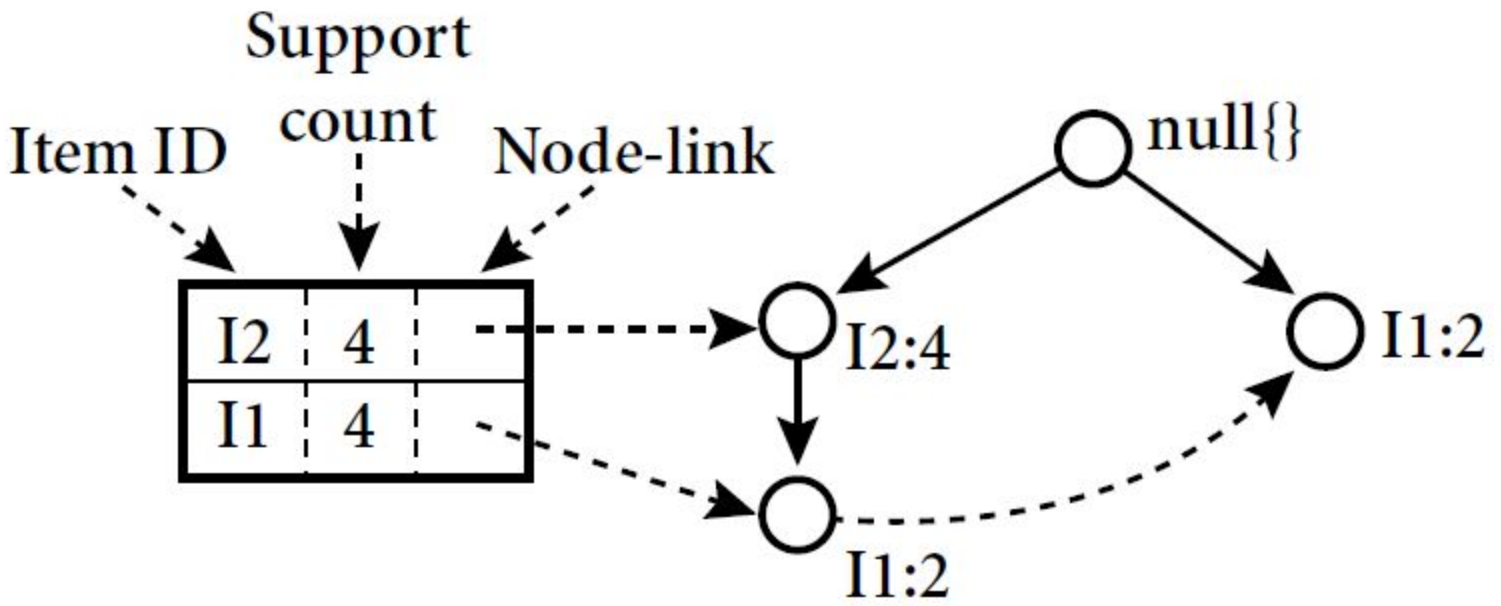


We have to start with item having minimum support count : I5

Item	*Conditional Pattern Base*	*Conditional FP-tree*	*Frequent Patterns Generated*
I5	{{I2, I1: 1}, {I2, I1, I3: 1}}	⟨I2: 2, I1: 2⟩	{I2, I5: 2}, {I1, I5: 2}, {I2, I1, I5: 2}
I4	{{I2, I1: 1}, {I2: 1}}	⟨I2: 2⟩	{I2, I4: 2}
I3	{{I2, I1: 2}, {I2: 2}, {I1: 2}}	⟨I2: 4, I1: 2⟩, ⟨I1: 2⟩	{I2, I3: 4}, {I1, I3: 4}, {I2, I1, I3: 2}
I1	{{I2: 4}}	⟨I2: 4⟩	{I2, I1: 4}

conditional FP-tree associated with the conditional node I3

Item	Conditional Pattern Base	Conditional FP-tree	Frequent Patterns Generated
I5	{{I2, I1: 1}, {I2, I1, I3: 1}}	⟨I2: 2, I1: 2⟩	{I2, I5: 2}, {I1, I5: 2}, {I2, I1, I5: 2}
I4	{{I2, I1: 1}, {I2: 1}}	⟨I2: 2⟩	{I2, I4: 2}
I3	{{I2, I1: 2}, {I2: 2}, {I1: 2}}	⟨I2: 4, I1: 2⟩, ⟨I1: 2⟩	{I2, I3: 4}, {I1, I3: 4}, {I2, I1, I3: 2}
I1	{{I2: 4}}	⟨I2: 4⟩	{I2, I1: 4}



FP-growth algorithm for discovering frequent itemsets without candidate generation

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 item 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 that $N.item\text{-}name = p.item\text{-}name$, 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*, *null*), which is implemented as follows.

```
procedure FP_growth(Tree, α)
    if Tree contains a single path P then
        for each combination (denoted as β) of the nodes in the path P
            generate pattern β ∪ α with support_count = minimum support count of nodes in β;
    else for each a_i in the header of Tree {
        generate pattern β = a_i ∪ α with support_count = a_i.support_count;
        construct β's conditional pattern base and then β's conditional FP_tree Tree_β;
        if Tree_β ≠ ∅ then
            call FP_growth(Tree_β, β); }
```

Advantages of the Pattern Growth Approach

- Divide-and-conquer:
  - Decompose both the mining task and DB according to the frequent patterns obtained so far
  - Lead 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

Vertical Data Format (ECLAT)

- Both the Apriori and FP-growth methods
  - *Horizontal data format*
  - *TID-itemset* format

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

- ECLAT (Equivalence CLASS Transformation algorithm)
  - developed by Zaki
  - Vertical data format
    - *item-TID set* format

itemset	*TID_set*
I1	{T100, T400, T500, T700, T800, T900}
I2	{T100, T200, T300, T400, T600, T800, T900}
I3	{T300, T500, T600, T700, T800, T900}
I4	{T200, T400}
I5	{T100, T800}

Mining frequent itemsets using vertical data format

- Intersecting the TID sets of every pair of frequent single items.
- minimum support count = 2

itemset	*TID_set*
I1	{T100, T400, T500, T700, T800, T900}
I2	{T100, T200, T300, T400, T600, T800, T900}
I3	{T300, T500, T600, T700, T800, T900}
I4	{T200, T400}
I5	{T100, T800}

Generated based on intersection of transactions

The 2-itemsets in vertical data format.

itemset	*TID_set*
{I1, I2}	{T100, T400, T800, T900}
{I1, I3}	{T500, T700, T800, T900}
{I1, I4}	{T400}
{I1, I5}	{T100, T800}
{I2, I3}	{T300, T600, T800, T900}
{I2, I4}	{T200, T400}
{I2, I5}	{T100, T800}
{I3, I5}	{T800}

Remove {I1,I4} and {I3,I5} as it does not full fill support count conditions

The 3-itemsets in vertical data format.

itemset	*TID_set*
{I1, I2, I3}	{T800, T900}
{I1, I2, I5}	{T100, T800}

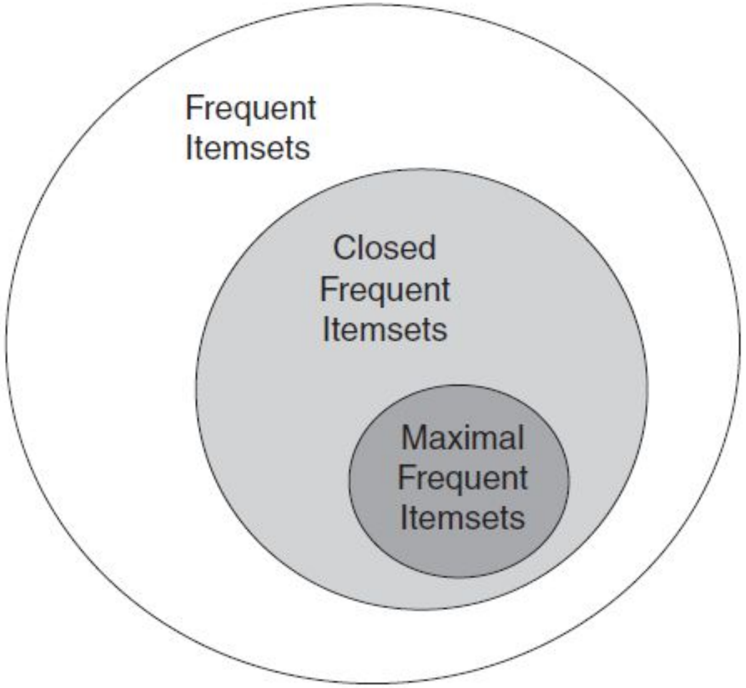
Generated based on intersection of transactions. The table displays only frequent itemsets. As other itemset like {I1,I3,I5}, {I2,I3,I4}, {I2,I3,I5}, {I2,I4,I5} are removed as it does not full fill support count conditions.

vertical data format

- **Advantages:**
  - no need to scan the database to find the support of (k+1) itemsets (for k>1)
- **Disadvantage:**
  - If, the TID sets can be quite long (Huge transaction database)
    - More space as well as computation time for intersecting the long sets
- **Improvement/solution for huge TID:**
- Diffset
  - {I1} = {T100, T400, T500, T700, T800, T900}
  - {I1, I2} = {T100,T400, T800, T900}.
  - *diffset*({I1, I2}, {I1}) = {T500, T700}

Closed Patterns and Max-Patterns

- A long pattern contains a combinatorial number of sub-patterns, e.g., $\{a_1, \ldots, a_{100}\}$ contains $({}_{100}{}^{1}) + ({}_{100}{}^{2}) + \ldots + ({}_{100}{}^{100}) = 2^{100} - 1 = 1.27 * 10^{30}$ sub-patterns!
- Solution: *Mine closed patterns and max-patterns instead*



Closed Patterns

- X is closed itemset
  - X is **frequent**
  - No immediate superset of X has same support as X

TID	Items
1	{A,B}
2	{B,C,D}
3	{A,B,C,D}
4	{A,B,D}
5	{A,B,C,D}

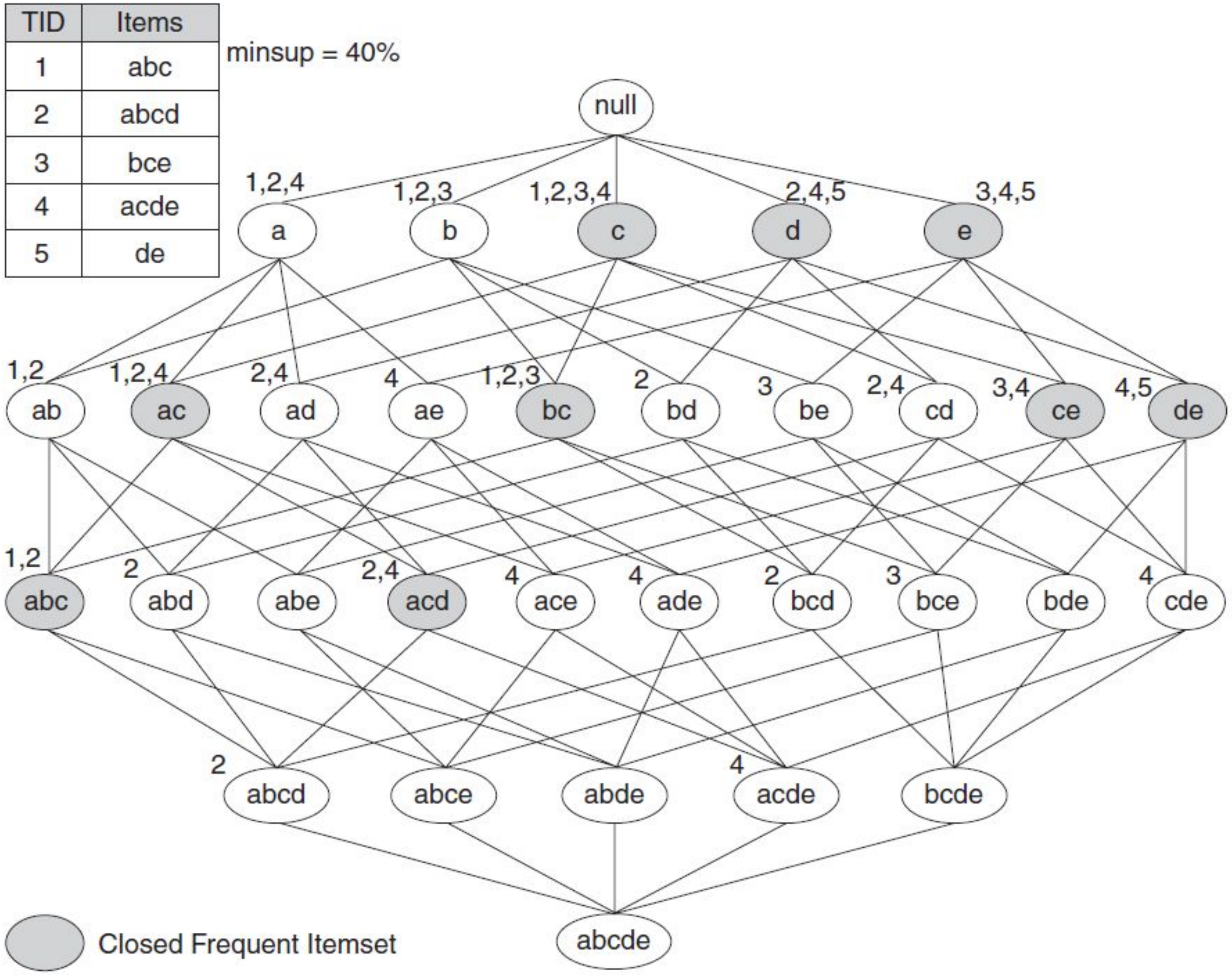
Itemset	Support
{A}	4
{B}	5
{C}	3
{D}	4
{A,B}	4
{A,C}	2
{A,D}	3
{B,C}	3
{B,D}	4
{C,D}	3

Itemset	Support
{A,B,C}	2
{A,B,D}	3
{A,C,D}	2
{B,C,D}	2
{A,B,C,D}	2

Closed Patterns

- X is closed itemset
  - X is **frequent**
  - No immediate superset of X has same support as X
  - For ex., ac has support of 3, the superset of ac : abc (2), acd(2) and ace (1) have support less than ac. Therefore ac is closed itemset.

TID	Items
1	abc
2	abcd
3	bce
4	acde
5	de

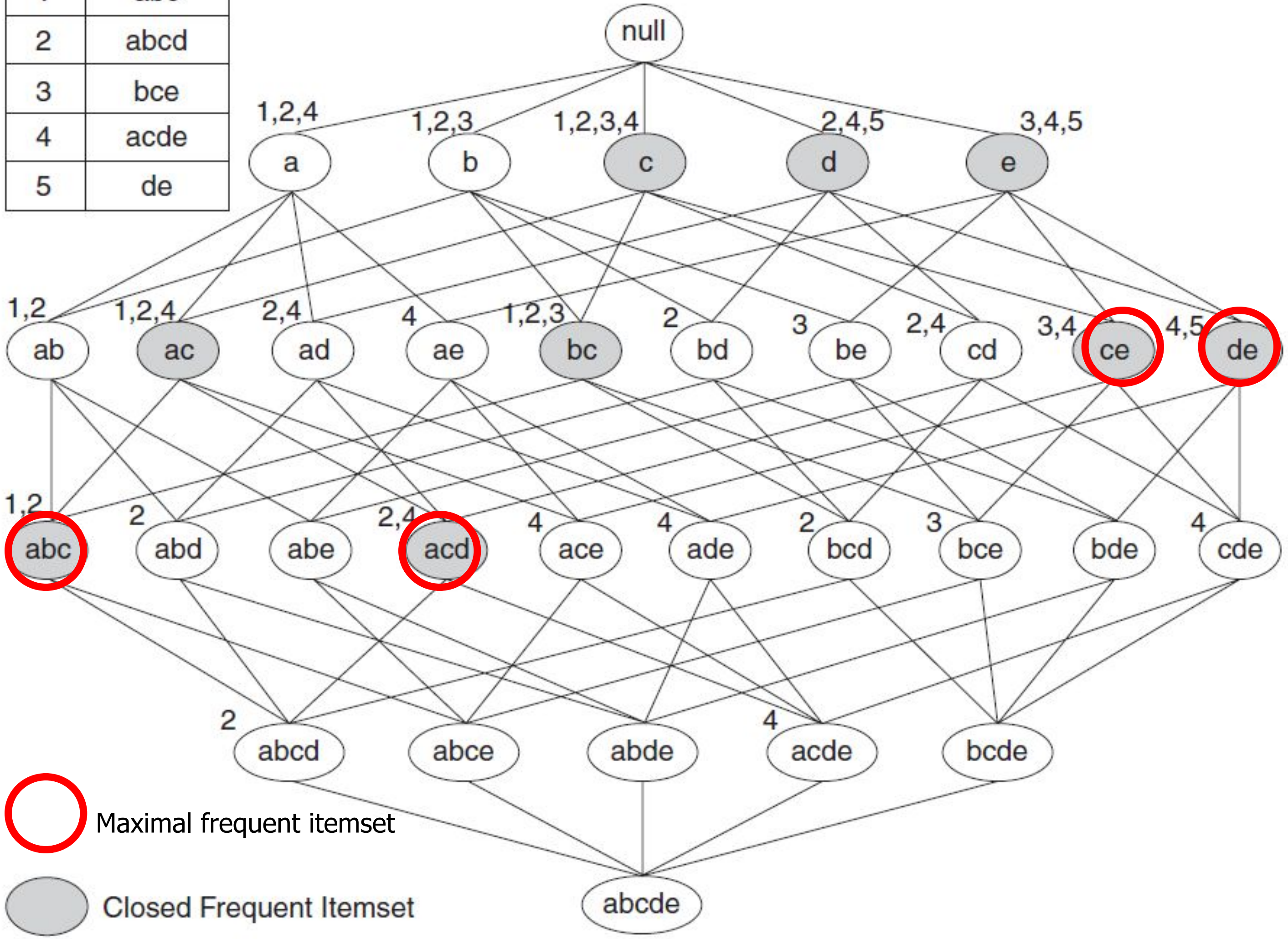


maximal frequent itemset

- X is a maximal frequent itemset
  - X is frequent
  - No immediate superset of X is frequent
  - Abc is maximum frequent (because a is frequent, its immediate superset ab is frequent, its immediate superset abc is frequent.)

TID	Items
1	abc
2	abcd
3	bce
4	acde
5	de

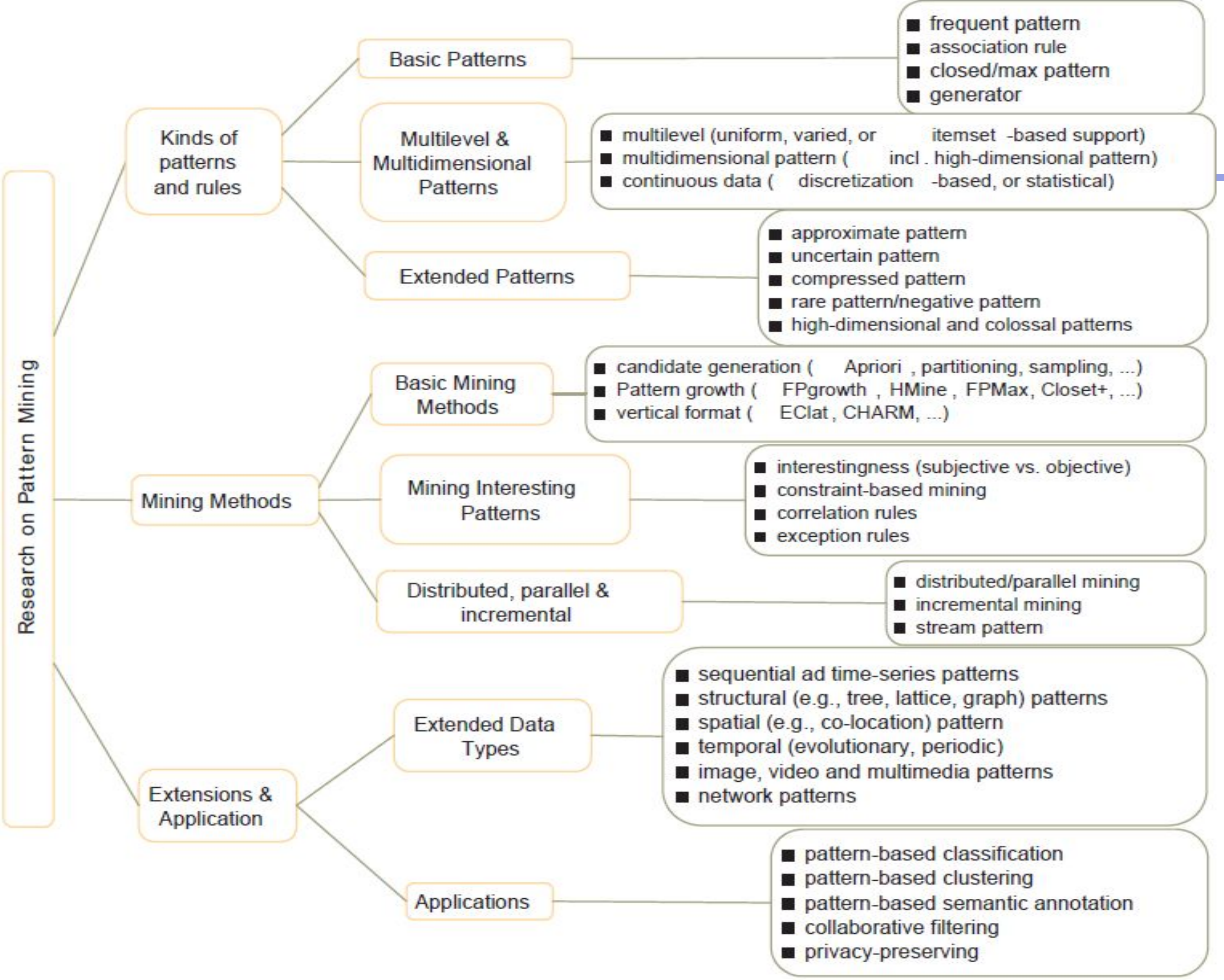
minsup = 40%



Closed Patterns and Max-Patterns

- Exercise. DB = {$<a_1, \ldots, a_{100}>$, $< a_1, \ldots, a_{50}>$}
  - Min_sup = 1.
- What is the set of all patterns?
  - $2^{100} - 1$
- What is the set of max-pattern?
  - $<a_1, \ldots, a_{100}>$: 1
- What is the set of closed itemset?
  - $<a_1, \ldots, a_{100}>$: 1
  - $< a_1, \ldots, a_{50}>$: 2

Research on Pattern Mining: A Road Map



Association Rules classification (continue..)

- **Based on the levels of abstractions involved in the rule**
  - **Single level association rule**
    - buys(X, "Computer") $\Rightarrow$ buys (X, "printer")
  - **Multilevel association rule**
    - Age(X,"30..39") $\Rightarrow$ buys (X, "computer")
      - Age(X,"30..39") $\Rightarrow$ buys (X, "laptop computer")
      - Age(X,"30..39") $\Rightarrow$ buys (X, "Desktop computer")

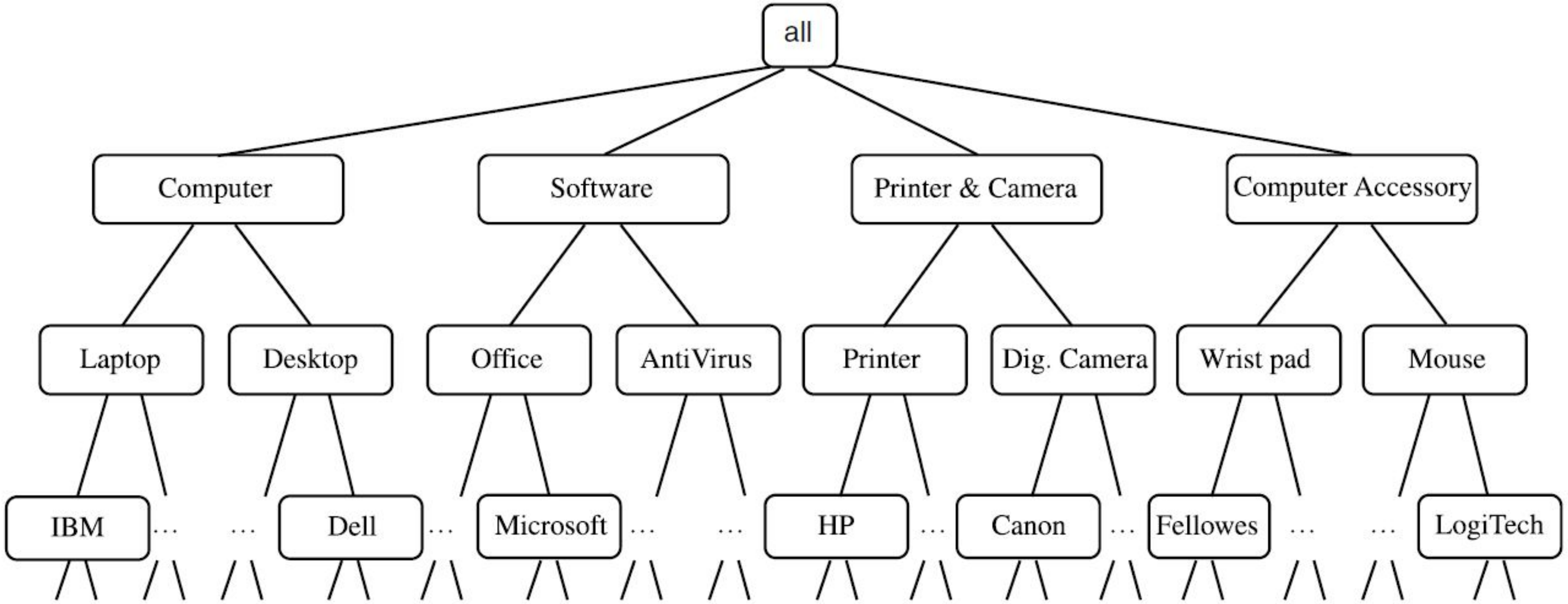
Multiple-level or Multilevel association

Lower Level:
"IBM-ThinkPad-R40/P4M" → "Symantec-Norton-Antivirus-2003"

Higher Level:
"IBM laptop computer" → "antivirus software"

Task-relevant data, *D*.

TID	*Items Purchased*
T100	IBM-ThinkPad-T40/2373, HP-Photosmart-7660
T200	Microsoft-Office-Professional-2003, Microsoft-Plus!-Digital-Media
T300	Logitech-MX700-Cordless-Mouse, Fellowes-Wrist-Rest
T400	Dell-Dimension-XPS, Canon-PowerShot-S400
T500	IBM-ThinkPad-R40/P4M, Symantec-Norton-Antivirus-2003
...	...

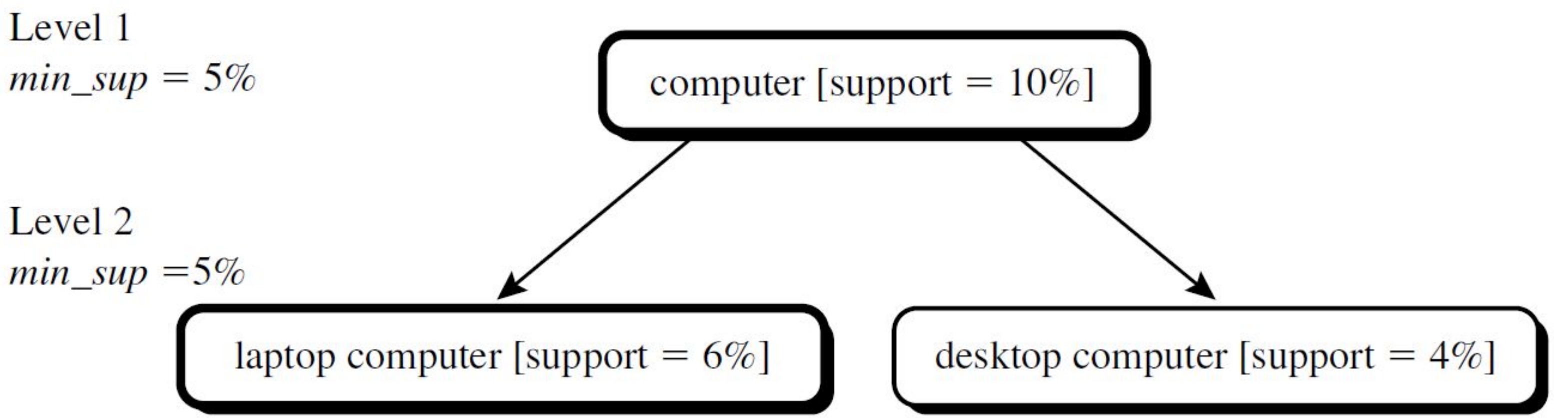


Multilevel association

- Based on support threshold
  - Using uniform minimum support for all levels (uniform support)
  - Using reduced minimum support at lower levels (reduced support)
  - Using item or group-based minimum support (group-based support)
    - user-specific, item, or group based minimal support thresholds

uniform support threshold

- Advantage:
  - Search procedure is simplified
  - Optimized by avoiding examining of descendants if ancestors does not satisfy the min_support
- Disadvantage:
  - Items at lower level may not be as frequent as higher level
    - If min_support is high then very few patterns at lower level
    - If min_support is low, then very large patterns at higher level

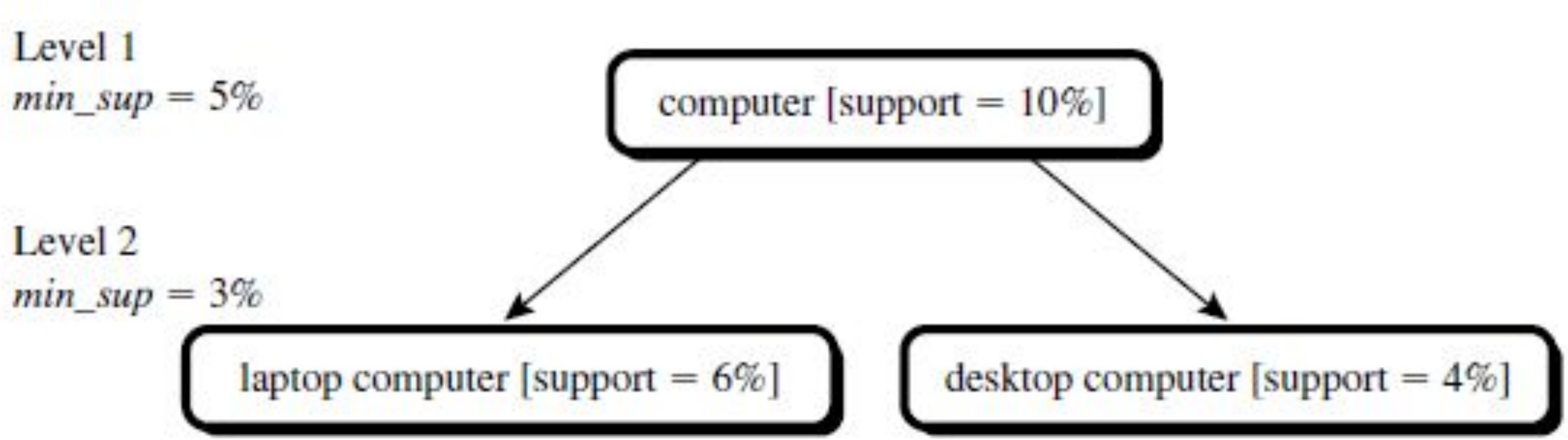


Reduced support threshold

- Level-by-level independent
- Level cross filtering by single item
- Level cross filtering by k-itemset

Reduced support threshold

- Level-by-level independent
  - Independent weather its parent node is frequent or not
  - Very relaxed as compared to other two.



Reduced support threshold

- Level cross filtering by single item
  - Item at i^{th} level is examined if its parent at $(i-1)^{th}$ level is frequent
  - may be miss few association between lower level items.
    - desktop computer => color monitor
    - Solution: level passage threshold

Level 1
Min_sup = 12%

computer [support = 10%]

Level 2
Min_sup = 3%

Laptop computer **(not examined)**

desktop computer**(not examined)**

Level 1
Min_sup = 12%

monitor [support = 10%]

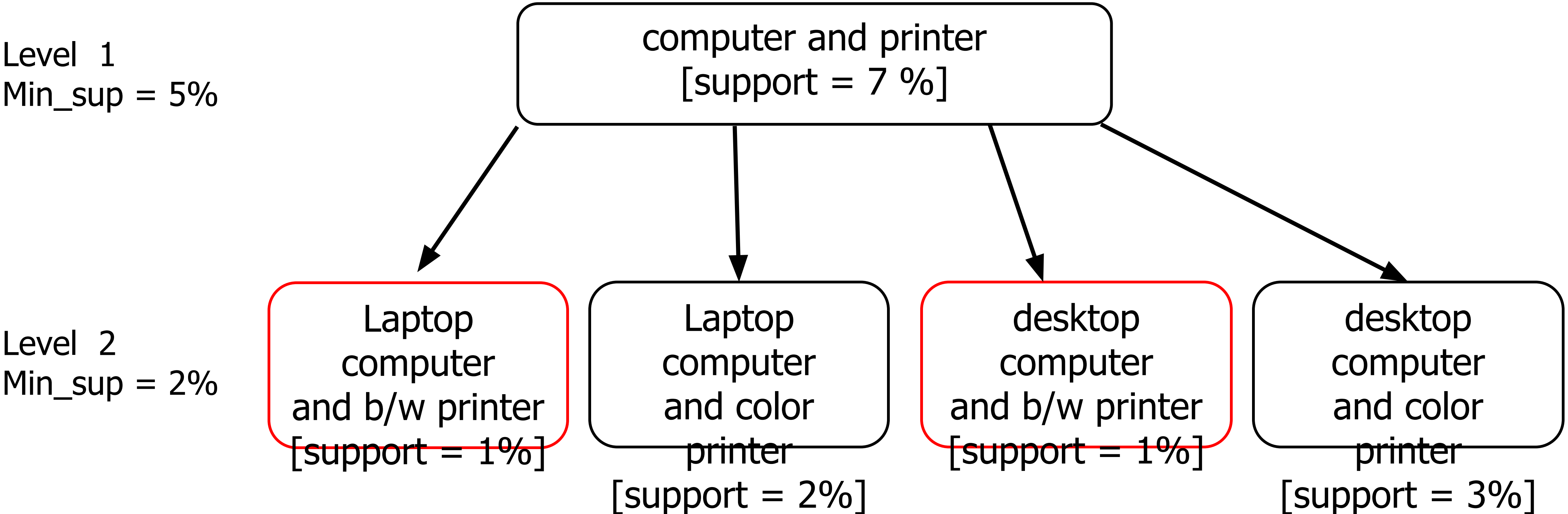
Level 2
Min_sup = 3%

b/w monitor **(not examined)**

color monitor **(not examined)**

Reduced support threshold

- Level cross filtering by k-itemset
  - A k-itemset at i^{th} level is examined if its corresponding parent k-itemset at $(i-1)^{th}$ level is frequent
  - Strong restriction that there are not many k-itemsets that when combined are also frequent

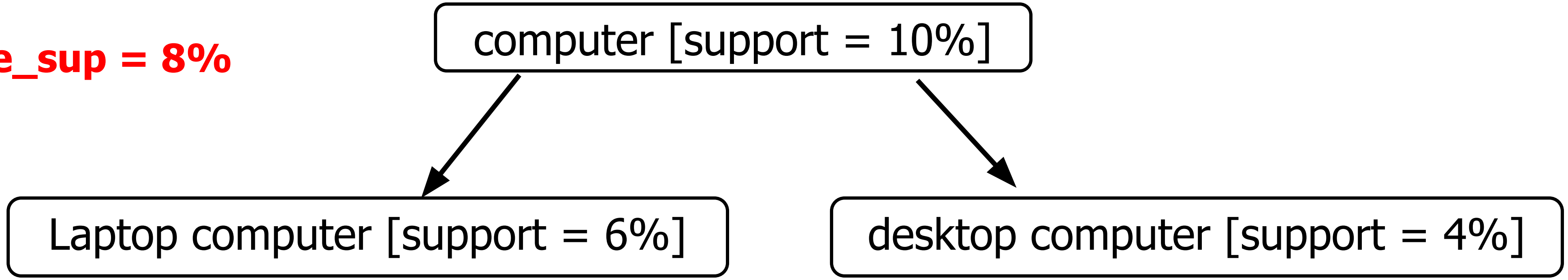


level passage threshold

- level passage threshold
  - Passing down relatively frequent items (sub-frequent items)
  - min_sup of lower level (i) < level passage threshold < min_sup of higher level (i-1)

Level 1
Min_sup = 12%
Level_passage_sup = 8%

Level 2
Min_sup = 3%



Cross-level association rules

- Same concept-level
  - Computer = > printer
  - Desktop computer => b/w printer

- Cross-level
  - Computer = > b/w printer
  - Lower conceptual level min_supp should be used

Challenge in Multilevel association

- Many **redundant rules** across multiple levels of abstraction due to the "ancestor" relationships among items.

$$buys(X, \text{"laptop computer"}) \Rightarrow buys(X, \text{"HP printer"})$$
$$[support = 8\%, confidence = 70\%]$$

$$buys(X, \text{"IBM laptop computer"}) \Rightarrow buys(X, \text{"HP printer"})$$
$$[support = 2\%, confidence = 72\%]$$

Mining Association Rules from Relational Databases and Data Warehouses

- **Based on the number of dimensions/predicate handled in the rule**
  - **Single dimensional association rule (intradimensional association rule)**
    - predicate occurs more than once within the rule
    - from transactional data
    - buys(X, "Computer") $\Longrightarrow$ buys (X, "printer")
  - **Multi dimensional association rule (interdimensional association rule)**
    - relational database or data warehouse
    - association rules containing multiple predicates
    - No repeated predicates
    - Age(X,"30..39") $\wedge$ income (X, "42K…48K) $\Longrightarrow$ buys (X, high resolution TV)
  - **hybrid-dimensional association rules**

$$age(X, \text{"}20...29\text{"}) \wedge buys(X, \text{"}laptop\text{"}) \Rightarrow buys(X, \text{"}HP\ printer\text{"})$$

Association Rules classification

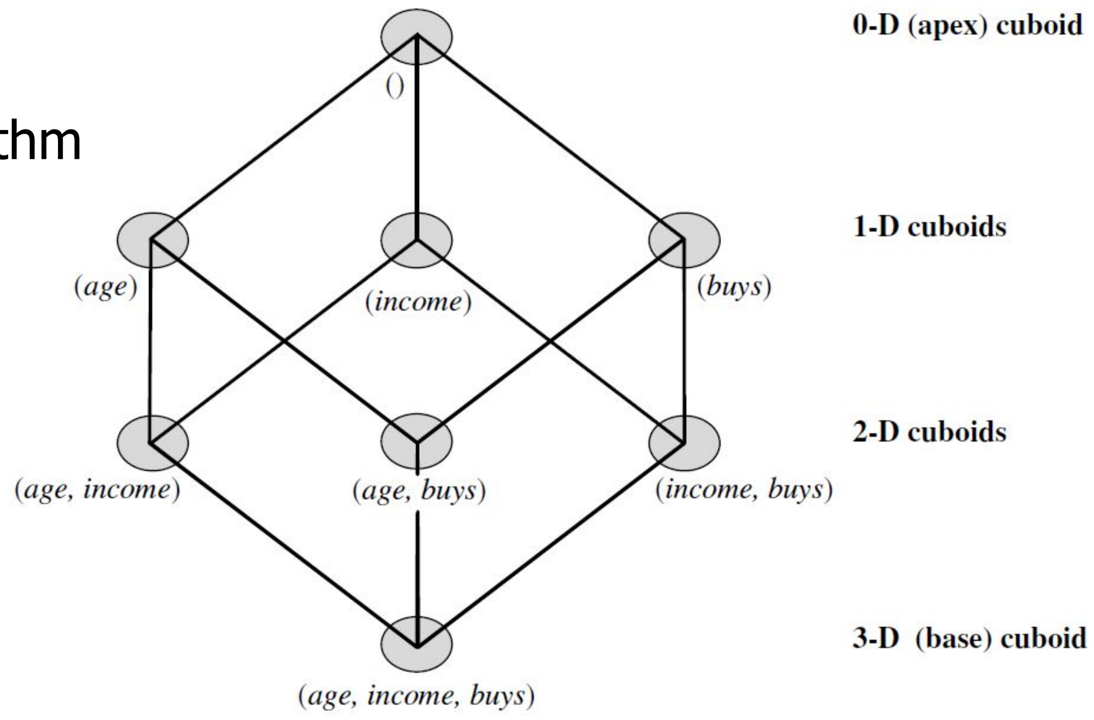
- **Based on the types of values handled in the rule**
  - Categorical attributes
    - occupation, brand, color
  - Quantitative attributes
    - *age, income, price*
  - Boolean association rule:
    - Presence and absence of item
    - buys(X, "Computer") $\Rightarrow$ buys (X, "printer")
  - Quantitative rule
    - Age(X,"30..39") $\wedge$ income (X, "42K…48K) $\Rightarrow$ buys (X, high resolution TV)

multidimensional association rules using static discretization of quantitative attributes

- Quantitative attributes are discretized prior to mining using predefined concept hierarchies
- E.g income: 0…20K, 21K…30K,31K…40K,41K…50K etc…
- **Categorical attributes** may also be generalized to higher concept hierarchy level

- Task relevant data stored **in relational table**
  - Treating each attribute-value pair as an itemset
  - Find all frequent predicate sets using any algorithm

- Task relevant data stored in **data cube**
  - Lattice of cuboid
  - Each cuboid correspond to the predicate sets and aggregated data



multidimensional association rules using dynamic discretization of quantitative attributes

- *dynamically* discretized during the mining process
- two-dimensional quantitative association rules

$$A_{quan1} \wedge A_{quan2} \Rightarrow A_{cat}$$

$$age(X, \text{“30...39”}) \wedge income(X, \text{“42K...48K”}) \Rightarrow buys(X, \text{“HDTV”})$$

- How to find such rules?
  - ARCS (Association Rule Clustering System)

ARCS (Association Rule Clustering System)

$A_{quan1} \wedge A_{quan2} \Rightarrow A_{cat}$ $\qquad$ $age(X, \text{“30...39”}) \wedge income(X, \text{“42K...48K”}) \Rightarrow buys(X, \text{“HDTV”})$

1. Maps pairs of quantitative attributes onto a 2-D grid for tuples satisfying a given categorical attribute condition
2. The grid is then searched for clusters of points from which the association rules are generated.

- Binning
  - Equal-width binning,
  - Equal-frequency binning,
  - Clustering-based binning
- ARCS uses equal-width binning

ARCS (Association Rule Clustering System)

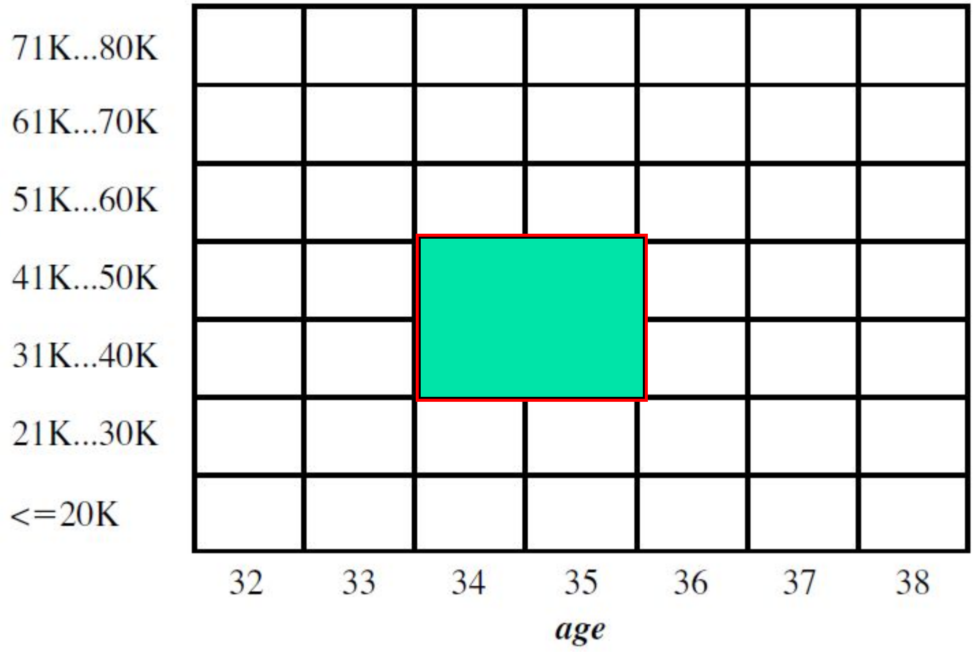
- A 2-D array is created
  - Age and income with all possible combination

1. **Finding frequent predicate sets** (find the frequent predicate sets those satisfying minimum support) and generate association rules that satisfying and minimum confidence.

$$age(X, 34) \wedge income(X, \text{“}31K...40K\text{”}) \Rightarrow buys(X, \text{“}HDTV\text{”})$$
$$age(X, 35) \wedge income(X, \text{“}31K...40K\text{”}) \Rightarrow buys(X, \text{“}HDTV\text{”})$$
$$age(X, 34) \wedge income(X, \text{“}41K...50K\text{”}) \Rightarrow buys(X, \text{“}HDTV\text{”})$$
$$age(X, 35) \wedge income(X, \text{“}41K...50K\text{”}) \Rightarrow buys(X, \text{“}HDTV\text{”}).$$

2. **Clustering the association rules**

$$age(X, \text{“}34...35\text{”}) \wedge income(X, \text{“}31K...50K\text{”}) \Rightarrow buys(X, \text{“}HDTV\text{”})$$



Interestingness Measure: Correlations (Lift)

- *play basketball* $\Rightarrow$ *eat cereal* [40%, 66.7%] is misleading
  - The overall % of students eating cereal is 75% > 66.7%.
- *play basketball* $\Rightarrow$ *not eat cereal* [20%, 33.3%] is more accurate, although with lower support and confidence
- Measure of dependent/correlated events: lift

$$lift = \frac{P(A \cup B)}{P(A)P(B)}$$

	Basketball	Not basketball	Sum (row)
Cereal	2000	1750	3750
Not cereal	1000	250	1250
Sum(col.)	3000	2000	5000

$$lift(B,C) = \frac{2000/5000}{3000/5000*3750/5000} = 0.89$$ <1 so, (Negative correlation)

$$lift(B,\neg C) = \frac{1000/5000}{3000/5000*1250/5000} = 1.33$$ >1 so, (Positive correlation)

If A corresponds to the sale of computer games and B corresponds to sale of videos, then the sale of games is said to “lift” the likelihood of the sale of videos by a factor of lift.

- Can not be identified by support-confidence framework

Correlation Analysis using χ^2

	game	$\overline{game}$	Σ_{row}
video	4,000 (4,500)	3,500 (3,000)	7,500
$\overline{video}$	2,000 (1,500)	500 (1,000)	2,500
Σ_{col}	6,000	4,000	10,000

Probability of purchasing a computer game is p({game}) = 0.6
Probability of purchasing a video is p({video}) = 0.75
Probability of purchasing both is p({game, video}) = 0.4

P({game, video}) / p({game}) X p({video}) = 0.4 / (0.6 X 0.75) = 0.89

- corr value is **less than one**, *buying game* and *buying video* are ***negatively correlated***.

Correlation Analysis using χ^2

	game	$\overline{game}$	Σ_{row}
video	4,000 (4,500)	3,500 (3,000)	7,500
$\overline{video}$	2,000 (1,500)	500 (1,000)	2,500
Σ_{col}	6,000	4,000	10,000

$$\chi^2 = \Sigma \frac{(observed\text{ - }expected)^2}{expected} = \frac{(4{,}000 - 4{,}500)^2}{4{,}500} + \frac{(3{,}500 - 3{,}000)^2}{3{,}000} + \frac{(2{,}000 - 1{,}500)^2}{1{,}500} + \frac{(500 - 1{,}000)^2}{1{,}000} = 555.6.$$

- χ^2 value is **greater than one**, and the observed value of the slot (*game*, *video*) = 4,000, which is less than the expected value 4,500, *buying game* and *buying video* are ***negatively correlated***.

Rule X ->Y Support, Confidence and lift

- Support
  - Freq(x,y)/N
  - Frequency of items bought over all transaction
- Confidence
  - freq(x,y)/freq(x)
  - Support(x,y)/support(x)
  - How often X and Y occurred together based on number of X occur (left side)
- Lift/correlation
  - How much more frequently the left-hand item is found with the right than without the right
  - Support(x,y)/support(x) * support (y)

Summary

- Basic concepts: association rules, support-confident framework, closed and max-patterns
- Scalable frequent pattern mining methods
  - Apriori (Candidate generation & test)
  - Projection-based (FPgrowth, CLOSET+, ...)
  - Vertical format approach (ECLAT, CHARM, ...)
- Association rule classification
- Which patterns are interesting?
  - Pattern evaluation methods