Association Rules Mining

Topics

- Basic concepts of Association Rules
- Rule strength measures
- Basic Algorithms
  - Apriori Algorithm
  - FP-Growth Algorithm
  - Other Approaches
  - Interestingness Measures
  - Sequential Pattern Mining
- Summary

Association rule mining

- Motivation: Finding inherent regularities in data
  - What products were often purchased together?— Clothes and Milk !
  - What are the subsequent purchases after buying a PC?
  - What kinds of DNA are sensitive to new drug?
  - Can we automatically recommend next web document?
- Applications
  - Basket data analysis, Cross-marketing, Rack arrangement, Sale campaign analysis
  - DNA sequence analysis
  - Web log (click stream) analysis

Association rule mining

- Frequent pattern
  - A pattern (a set of items, subsequences, substructures, etc.) that occurs frequently in a data set
- First proposed by Agrawal et al. in 1993 in the context of frequent itemsets and association rule mining
- An important data mining model studied extensively

Association rule mining

- Given a set of transactions, find rules that will predict the occurrence of an item based on the occurrences of other items in the transaction
- Initially used for Market Basket Analysis to find how items purchased by customers are related

  Bread → Milk [Sup = 5%, Conf = 100%]

The model: Data

- $I = \{i_1, i_2, \ldots, i_m\}$: a set of *items*
- Transaction t: a set of items, and $t \subseteq I$
- Transaction Database T: a set of transactions $T = \{t_1, t_2, \ldots, t_n\}$

Transaction data: Supermarket data

- Market basket transactions:

  t1: {bread, cheese, milk}

  t2: {apple, biscuit, salt, yogurt}

  … …

  tn: {biscuit, bread, milk}

- Concepts:
  - An *item*: an item/article in a basket
  - *I*: the set of all items sold in the store
  - A *transaction*: items purchased in a basket; it may have TID (transaction ID)
  - A *transactional dataset*: A set of transactions

Transaction data: a set of documents

- **Text document data set, each document is treated as a "bag" of keywords**

  doc1: Student, Teach, School
  doc2: Student, School
  doc3: Teach, School, City, Game
  doc4: Baseball, Basketball
  doc5: Basketball, Player, Spectator
  doc6: Baseball, Coach, Game, Team
  doc7: Basketball, Team, City, Game

- **Web page data set**

  Session1: PageA.html, PageB.html, PageC.html
  Session2: PageC.html, PageD.html, PageE.html
  Session3: PageA.html, PageC.html, PageD.html

The model: Rules

- A transaction *t* contains *X*, a set of items (itemset) in *I*, if $X \subseteq t$
- An association rule is an implication of the form:

  $X \rightarrow Y$, where $X, Y \subset I$, *and* $X \cap Y = \varnothing$

- An itemset is a set of items
  - E.g., X = {milk, bread, cereal} is an itemset
- A *k*-itemset is an itemset with *k* items
  - E.g., {milk, bread} is a 2-itemset

  {milk, bread, cereal} is a 3-itemset

Rule Strength Measures

- An association rule is a pattern that states when *X* occurs, *Y* occurs with certain probability
  - Support
  - Confidence

Support and Confidence

- Support
  - The rule holds with support *sup* in *T* (the transaction data set having n transactions) if sup% of transactions contain $X \cup Y$
  - *sup* = Pr($X \cup Y$)

$$\text{sup} = \frac{(X \cup Y).count}{n}$$

    - Relative Support
  - The frequency count of an itemset $X \cup Y$, denoted by (*XUY).count*, in a data set *T* is the number of transactions
    - Count/Absolute Support

Rule strength measures

- Confidence
  - The rule holds in *T* with confidence *conf* if % of transactions that contain *X* also contain *Y.*
  - *conf* = Pr(*Y* | *X*)

$$confidence = \frac{(X \cup Y).count}{X.count}$$

Goal and key features

- **Goal:** Find all rules that satisfy the user-specified *minimum support* (minsup) and *minimum confidence* (minconf)
- **Key Features**
  - Completeness: find all rules
  - Compute the support and confidence for each rule
  - Prune rules that fail the *minsup* and *minconf* thresholds
  - Mining with data on hard disk (not in memory)

An example

t1: Bread, Biscuit, Milk
t2: Bread, Cheese
t3: Cheese, Boots
t4: Bread, Biscuit, Cheese
t5: Bread, Biscuit, Clothes, Cheese, Milk
t6: Biscuit, Clothes, Milk
t7: Biscuit, Milk, Clothes

- Transaction data
- Assume:
  minsup = 30%
  minconf = 80%
- An example frequent *itemset*: {Biscuit, Clothes, Milk} [sup = 3/7]
- Association rules from the itemset:
  Clothes → Milk, Biscuit [sup = 3/7, conf = 3/3]
  … …
  Clothes, Biscuit → Milk, [sup = 3/7, conf = 3/3]

Assumption

- A simplistic view of shopping baskets transactions
  - Some important information not considered e.g.
    - The quantity of each item purchased
    - The price paid
- Assume all data are categorical
  - Examples:
    - Item Purchased or not ?
    - ID numbers, eye color {brown, black, etc.}, zip codes
    - Height in {tall, medium, short}

Many mining algorithms

- A large number of them!!
- Use of different strategies and data structures
- Resulting sets of rules are all the same
- Computational efficiencies and memory requirements may be different

The Apriori algorithm

- **The best known algorithm**
- **Two steps**:
  - Find all itemsets that have minimum support (*frequent itemsets*, also called large itemsets)
  - Use frequent itemsets to generate rules

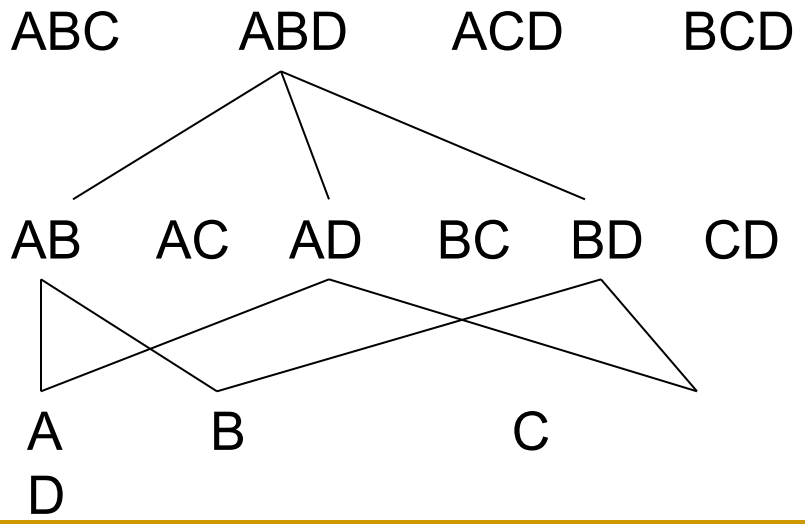
- E.g., a frequent itemset
  {Biscuit, Clothes, Milk} [sup = 3/7]
  and one rule from the frequent itemset
  Clothes → Milk, Biscuit [sup = 3/7, conf = 3/3]

Step 1: Mining all frequent itemsets

- A frequent *itemset* is an itemset whose support is ≥ minsup
- Key idea
  - The apriori property (downward closure property)
    - Any subsets of a frequent itemset are also frequent itemsets

If **{juice, glass, nuts}** is frequent, so is **{juice, glass}**

i.e., every transaction having {juice, glass, nuts} also contains {juice, glass}



The Algorithm

- Iterative algo. (also called level-wise search): Find all 1-item frequent itemsets; then all 2-item frequent itemsets, and so on
  - In each iteration *k*, only consider itemsets that contain some *k*-1 frequent itemset

- Find frequent itemsets of size 1: F_1
- For $k = 2$
  - C_k = candidates of size *k*: those itemsets of size *k* that could be frequent, given F_{k-1}
  - F_k = those itemsets that are actually frequent, $F_k \subseteq C_k$ (need to scan the database once)

The Apriori Algorithm—An Example

Database T

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

$Sup_{min} = 2$

The Apriori Algorithm—An Example

$Sup_{min} = 2$

Database T

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

C_1

1st scan →

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

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

C_1

1st scan

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

L_1

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

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

C_1

1st scan

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}

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

C_1

1st scan

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}

2nd scan

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

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

C_1

1st scan

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

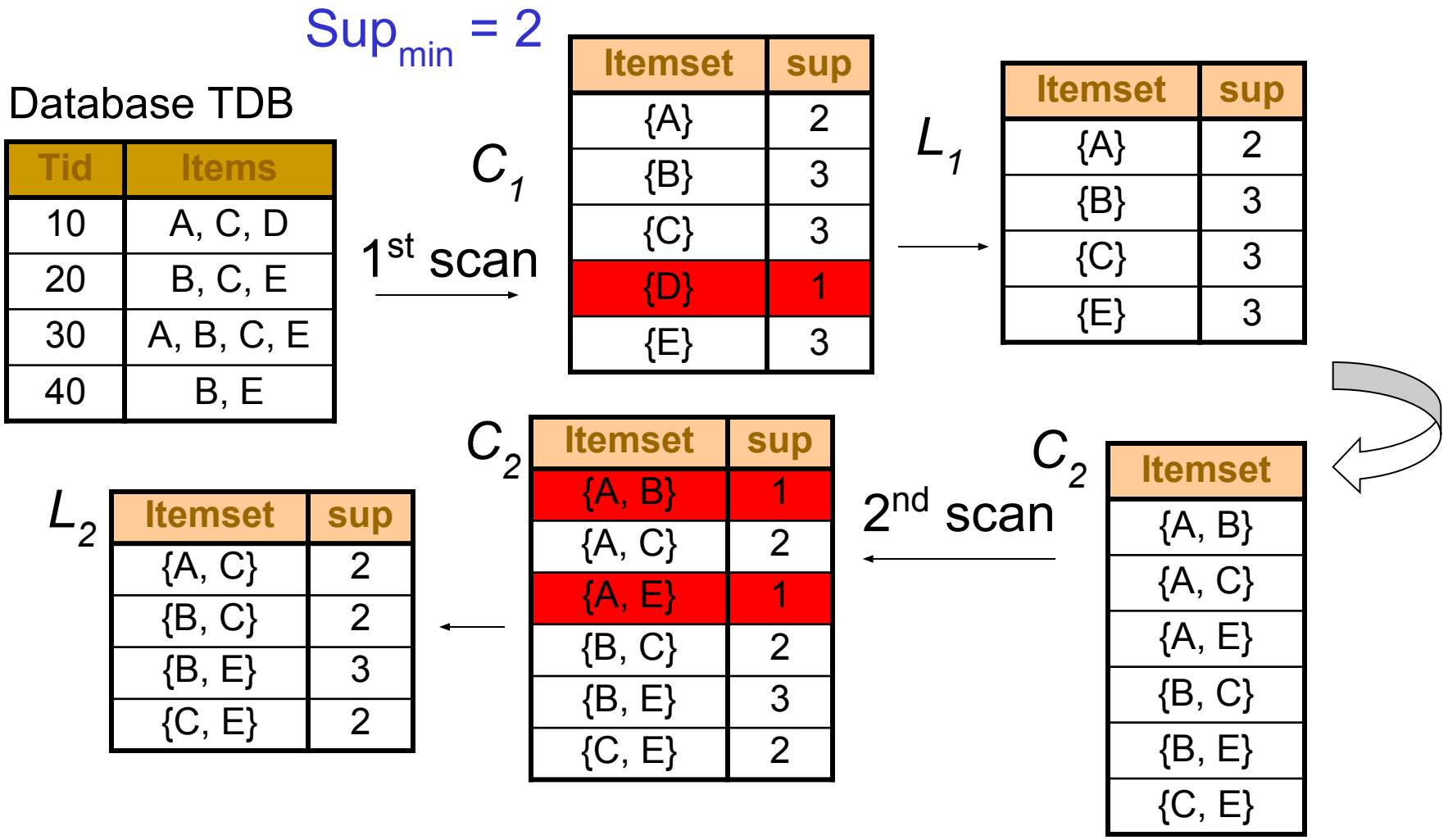
Itemset
{A, B}
{A, C}
{A, E}
{B, C}
{B, E}
{C, E}

2nd scan

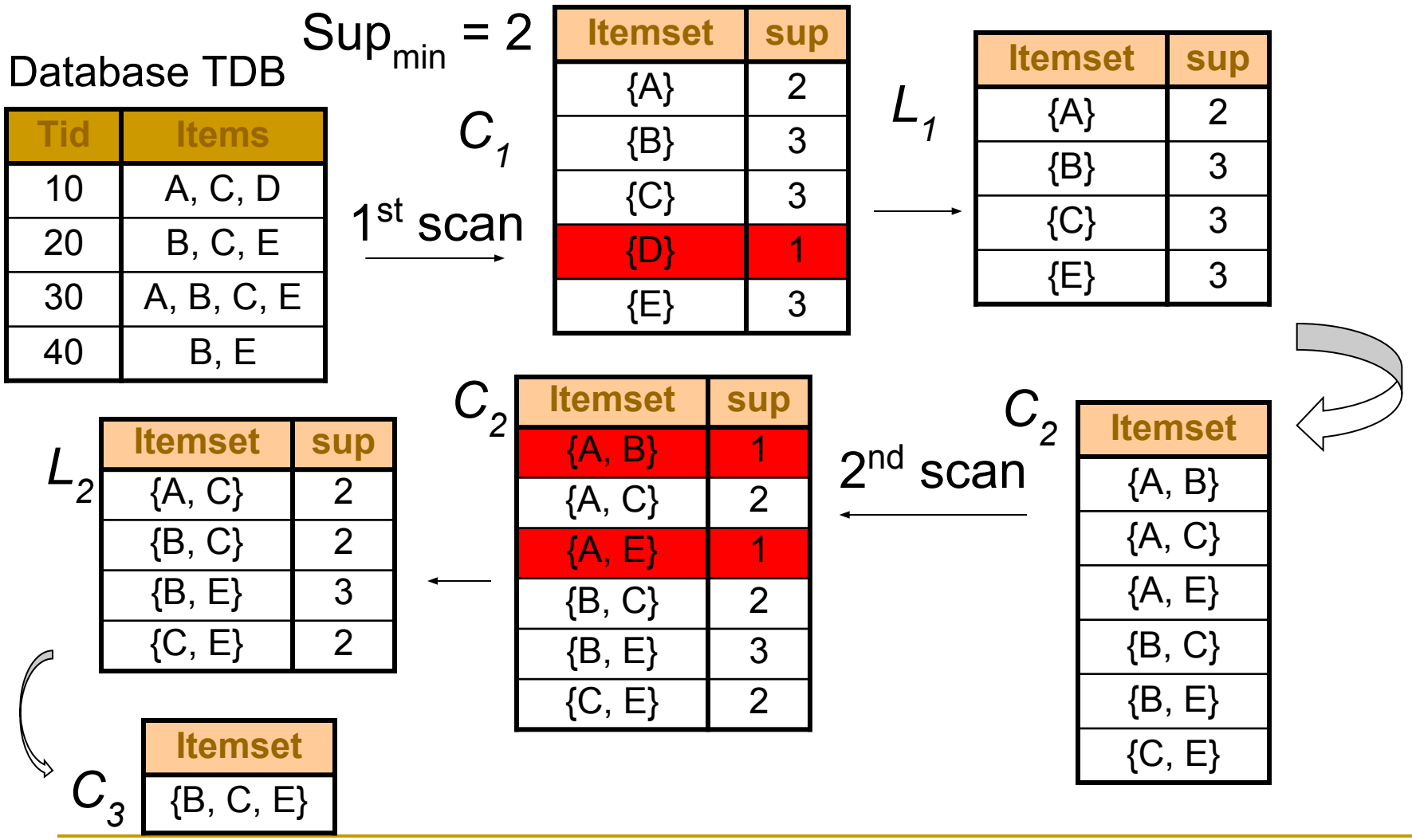
C_2

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

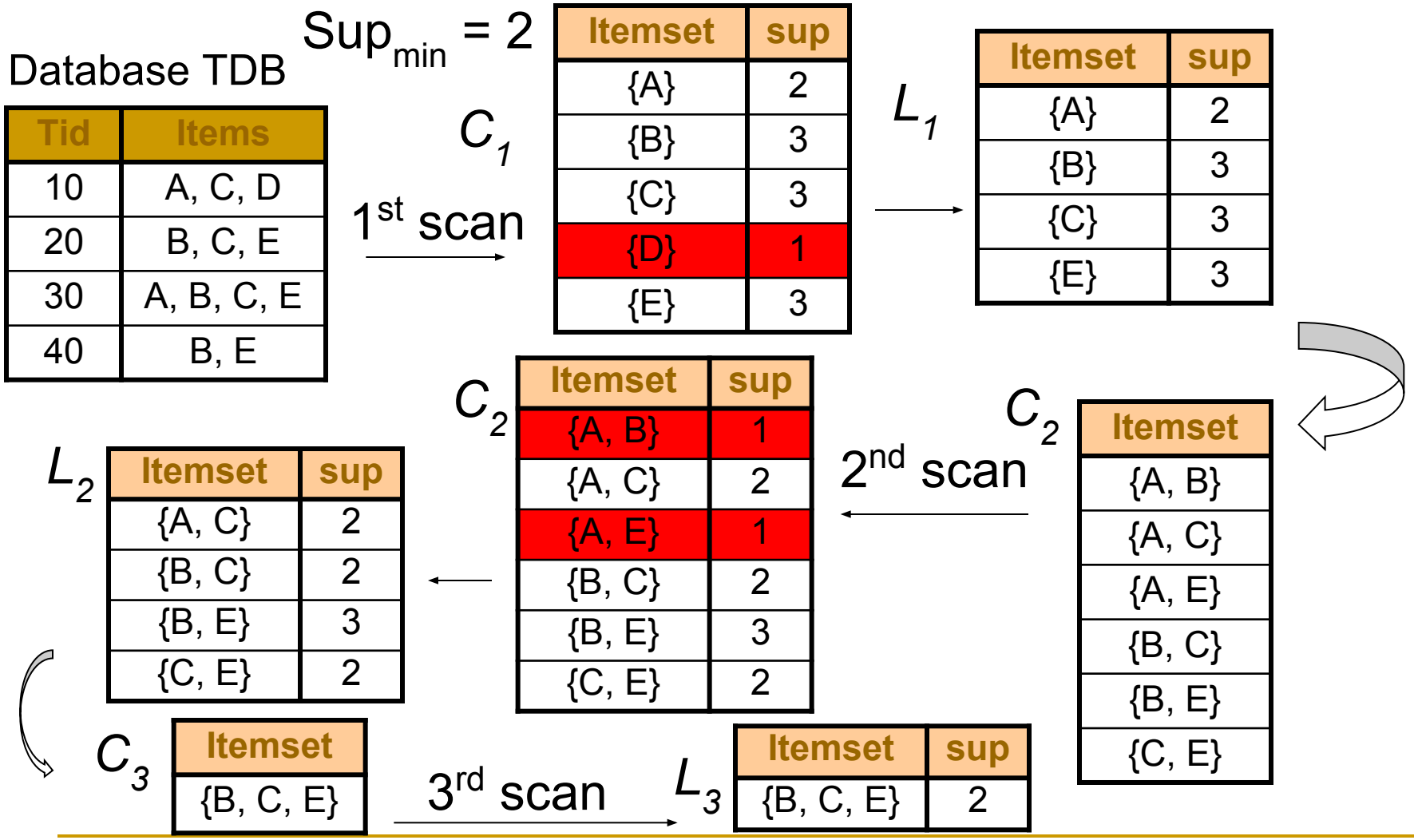
The Apriori Algorithm—An Example



The Apriori Algorithm—An Example



The Apriori Algorithm—An Example



The Apriori Algorithm

C_k: Candidate itemset of size k

F_k : frequent itemset of size k

Algorithm Apriori(*T*)

```
    C1 ← init-pass(T);
    F1 ← {f | f ∈ C1, f.count/n ≥ minsup};    // n: no. of transactions in T
    for (k = 2; Fk-1 ≠ ∅; k++) do
      Ck ← candidate-gen(Fk-1);
      for each transaction t ∈ T do
          for each candidate c ∈ Ck do
            if c is contained in t then
              c.count++;
          end
      end
          Fk ← {c ∈ Ck | c.count/n ≥ minsup}
    end
return F ← Uk Fk;
```

Apriori candidate generation

- Function takes F_{k-1} and returns a superset (called the candidates) of the set of all frequent k-itemsets
- It has two steps
  - *join* step: Generate all possible candidate itemsets C_k of length k
  - *prune* step: Remove those candidates in C_k that cannot be frequent

Implementation of Apriori

- 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*}

Assignment example

1. **2-Itemset=**{{A, C}, {B, C}, {B, E}, {C, E}}
   - **3-itemset ?**

2. **2-Itemset=**{{I1,I2}, {I1,I3}, {I1,I5}, {I2,I3}, {I2,I4}, {I2,I5}}}
   - **3-itemset ?**

Candidate-gen function

Function candidate-gen(F_{k-1})

$C_k \leftarrow \varnothing$;

forall $f_1, f_2 \in F_{k-1}$

with $f_1 = \{i_1, \ldots, i_{k-2}, i_{k-1}\}$

and $f_2 = \{i_1, \ldots, i_{k-2}, i'_{k-1}\}$

and $i_{k-1} < i'_{k-1}$ **do**

$c \leftarrow \{i_1, \ldots, i_{k-1}, i'_{k-1}\}$; // join f_1 and f_2

$C_k \leftarrow C_k \cup \{c\}$;

for each (k-1)-subset s of c **do**

if ($s \notin F_{k-1}$) **then**

delete c from C_k; // prune

end

end

return C_k;

Step 2: Generating rules from frequent itemsets

- Frequent itemsets ≠ association rules
- For each frequent itemset *X*,

  For each proper nonempty subset *A* of *X*,
  - Let *B* = X - *A*
  - A ⟶ B is an association rule if
    - Confidence(A ⟶ B) ≥ minconf,

      support(A ⟶ B) = support(A ∪ B) = support(X)

      confidence(A ⟶ B) = support(A ∪ B) / support(A)

Generating Rules: an example

- Suppose {2,3,4} is frequent, with sup=50%
  - Proper nonempty subsets: {2,3}, {2,4}, {3,4}, {2}, {3}, {4}, with sup=50%, 50%, 75%, 75%, 75%, 75% respectively
  - These generate these association rules:
    - 2,3 → 4 confidence=100%
    - 2,4 → 3 confidence=100%
    - 3,4 → 2 confidence=67%
    - 2 → 3,4 confidence=67%
    - 3 → 2,4 confidence=67%
    - 4 → 2,3 confidence=67%
    - All rules have support = 50%

Generating Rules: summary

- To recap, in order to obtain A → B, we need to have support(A ∪ B) and support(A)
- All the required information for confidence computation has already been recorded in itemset generation
  - No need to see the data *T* any more
- This step is not as time-consuming as frequent itemsets generation

Assignment Exercise: 1

- A database has five transactions.

Let *min sup* = 60% and *min con f* = 80%.

TID	***items bought***
T100	{M, O, N, K, E, Y}
T200	{D, O, N, K, E, Y}
T300	{M, A, K, E}
T400	{M, U, C, K, Y}
T500	{C, O, O, K, I ,E}

Find all frequent itemsets using Apriori.

Apriori Algorithm

Seems to be very expensive

- Breadth-first (Level-wise) search
- If, K = the size of the largest itemset then makes at most K passes over data
- Very simple and fast
  - Under some conditions, all rules can be found in linear time
- Scale up to large data sets

Apriori Algorithm

- Major computational challenges
  - Multiple scans of transaction database
  - Huge number of candidates
    - The number of frequent itemsets to be generated is senstive to the minsup threshold
    - When minsup is low, there exist potentially an exponential number of frequent itemsets
    - Example:
      - 10^4 frequent 1-itemsets, generate more than 10^7 candidate 2-itemsets
      - To discover a frequent pattern of size 100, such as $\{a_1, \ldots, a_{100}\}$
        - Generated candidates $2^{100} - 1$ = (Approx.) 10^{30}
  - Tedious workload of support counting for candidates

Apriori Algorithm

- Improving Apriori: general ideas
  - Reduce passes of transaction database scans
  - Shrink number of candidates
  - Facilitate support counting of candidates

Mining Frequent Patterns without Candidate Generation ???

Pattern-Growth Approach: Mining Frequent Patterns Without Candidate Generation

- The FPGrowth Approach given by J. Han, J. Pei, and Y. Yin, SIGMOD' 00
  - Depth-first search
  - Avoid explicit candidate generation

FPGrowth Approach

- 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
- An efficient, FP-tree-based frequent pattern mining method
  - A divide-and-conquer methodology: decompose mining tasks into smaller ones calls conditional databases
  - Avoid candidate generation: sub-database mining only!

Example

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

min_support = 3

Step 1: Scan DB once, find frequent 1-itemset (single item pattern)

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

min_support = 3

Header Table

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

Step 2: Sort frequent items in frequency descending order, f-list

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

min_support = 3

Header Table

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

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

Step 3: Scan DB again, construct FP-tree

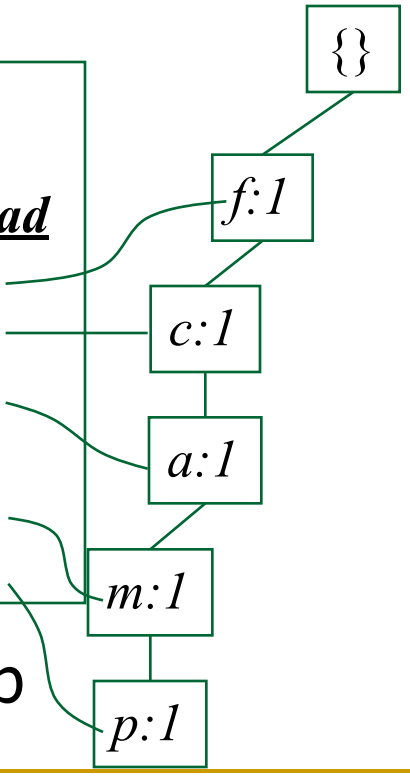
TID	Items bought	(ordered) frequent items
100	{f, a, c, d, g, i, m, p}	{f, c, a, m, p}
200	{a, b, c, f, l, m, o}	{f, c, a, b, m}
300	{b, f, h, j, o, w}	{f, b}
400	{b, c, k, s, p}	{c, b, p}
500	{a, f, c, e, l, p, m, n}	{f, c, a, m, p}

***min_support* = 3**

- To facilitate the tree traversal, an item header table is built with a chain of node-links

Header Table

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



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

Step 3: Cont…

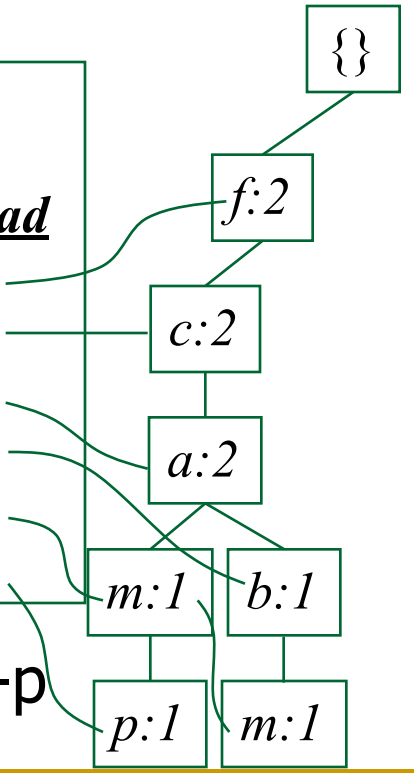
TID	Items bought	(ordered) frequent items
100	{f, a, c, d, g, i, m, p}	{f, c, a, m, p}
200	{a, b, c, f, l, m, o}	{f, c, a, b, m}
300	{b, f, h, j, o, w}	{f, b}
400	{b, c, k, s, p}	{c, b, p}
500	{a, f, c, e, l, p, m, n}	{f, c, a, m, p}

min_support = 3

- To facilitate the tree traversal, an item header table is built with a chain of node-links

Header Table

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



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

Step 3: Cont…

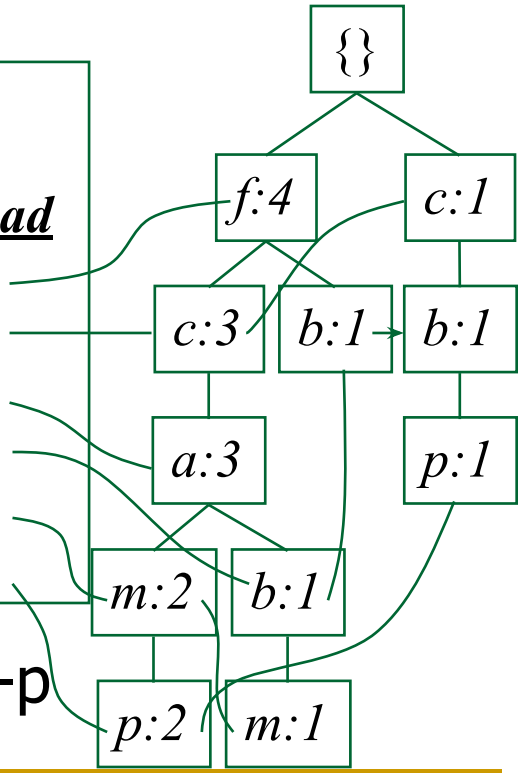
TID	Items bought	(ordered) frequent items
100	{f, a, c, d, g, i, m, p}	{f, c, a, m, p}
200	{a, b, c, f, l, m, o}	{f, c, a, b, m}
300	{b, f, h, j, o, w}	{f, b}
400	{b, c, k, s, p}	{c, b, p}
500	{a, f, c, e, l, p, m, n}	{f, c, a, m, p}

***min_support* = 3**

- To facilitate the tree traversal, an item header table is built with a chain of node-links

Header Table

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



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

FPGrowth Example

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

MinSup=2

FPGrowth Assignment-2

Prepare the FP-Tree

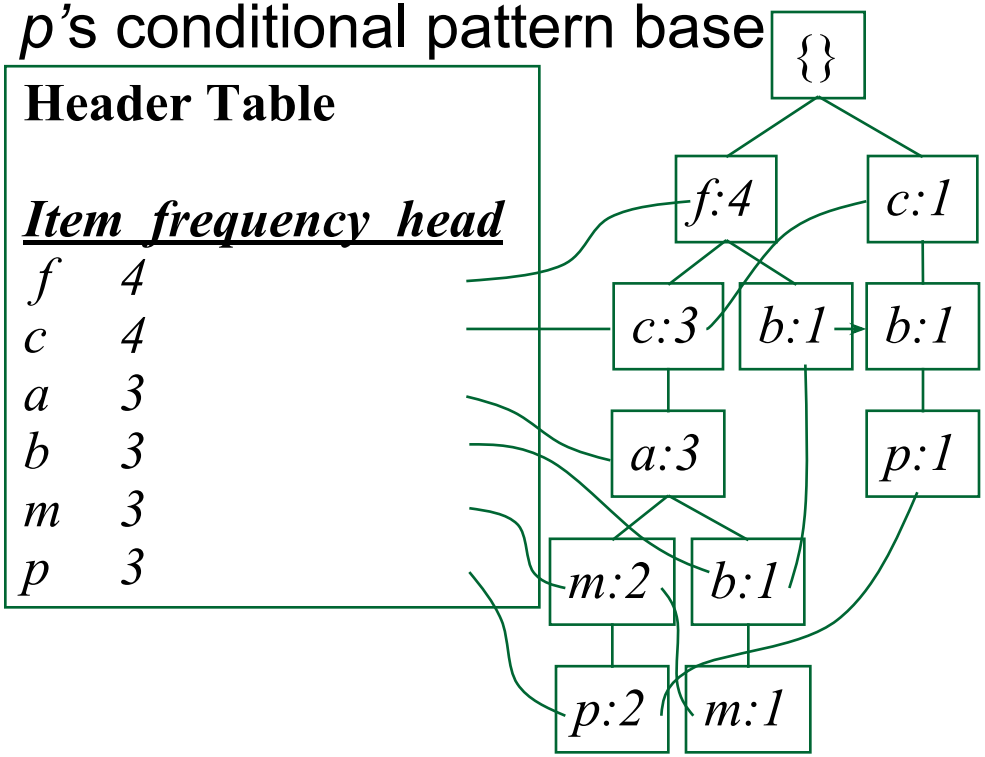
TID	Items
1	{A,B}
2	{B,C,D}
3	{A,C,D,E}
4	{A,D,E}
5	{A,B,C}
6	{A,B,C,D}
7	{B,C}
8	{A,B,C}
9	{A,B,D}
10	{B,C,E}

Step 4: Mining of FP-Tree: Partition Patterns and Databases

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

Find Patterns Having P From P-conditional Database

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

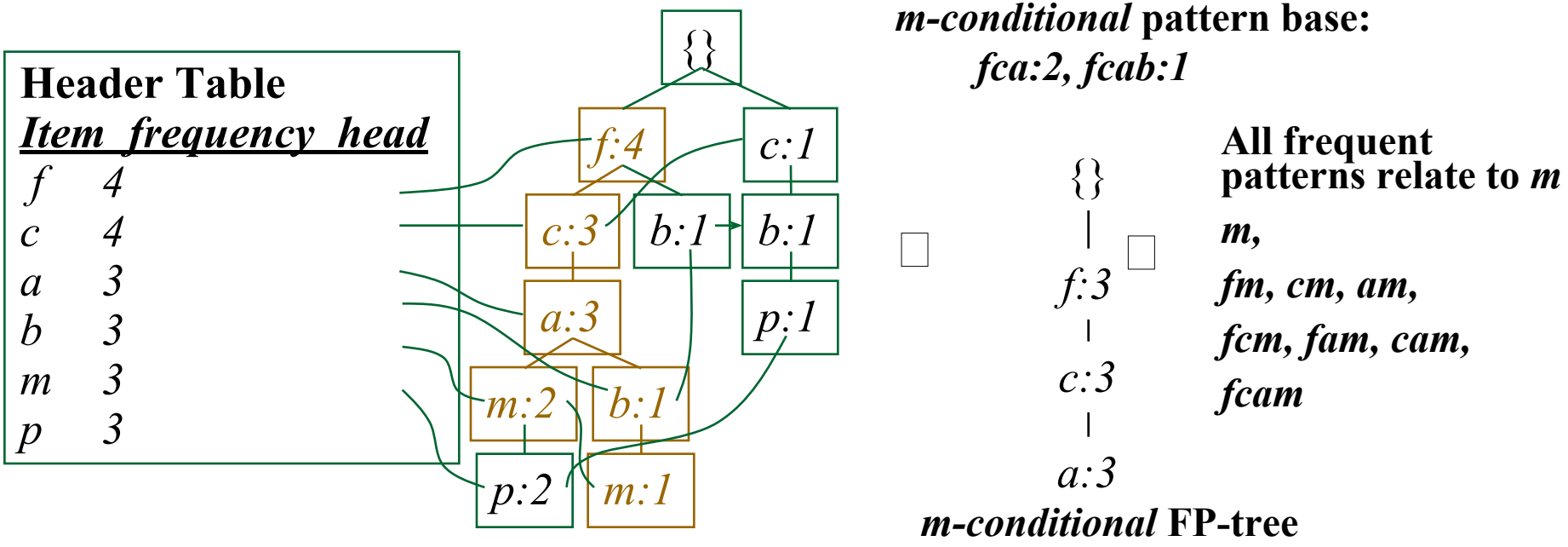


***Conditional* pattern bases**

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

From Conditional Pattern-bases to Conditional FP-trees

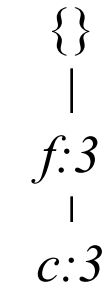
- For each pattern-base
  - Accumulate the count for each item in the base
  - Construct the FP-tree for the frequent items of the pattern base
    - having support count greater than the min support



- Conditional FP-Tree: Including items having support count greater than the min support

Recursion: Mining Each Conditional FP-tree

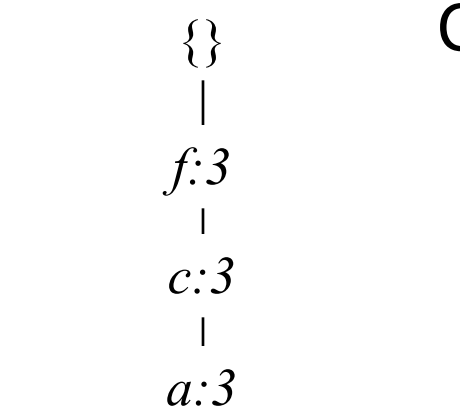
Cond. pattern base of "am": (fc:3)



***am-conditional* FP-tree**

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

***cm-conditional* FP-tree**



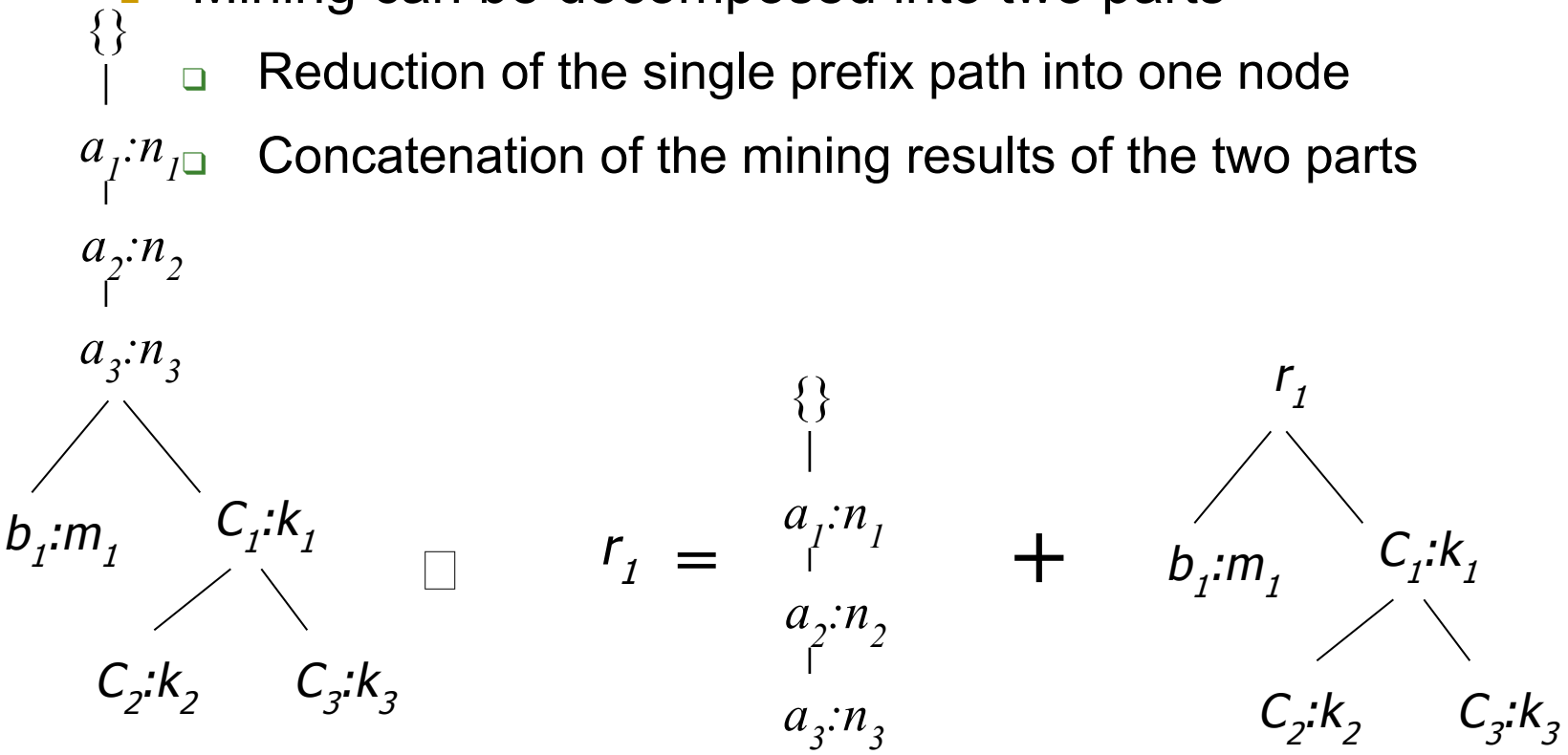
***m-conditional* FP-tree**

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

***cam-conditional* FP-tree**

A Special Case: Single Prefix Path in FP-tree

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



The FP-Growth Mining Method

- 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

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.

FP-Growth Algorithm Cont…

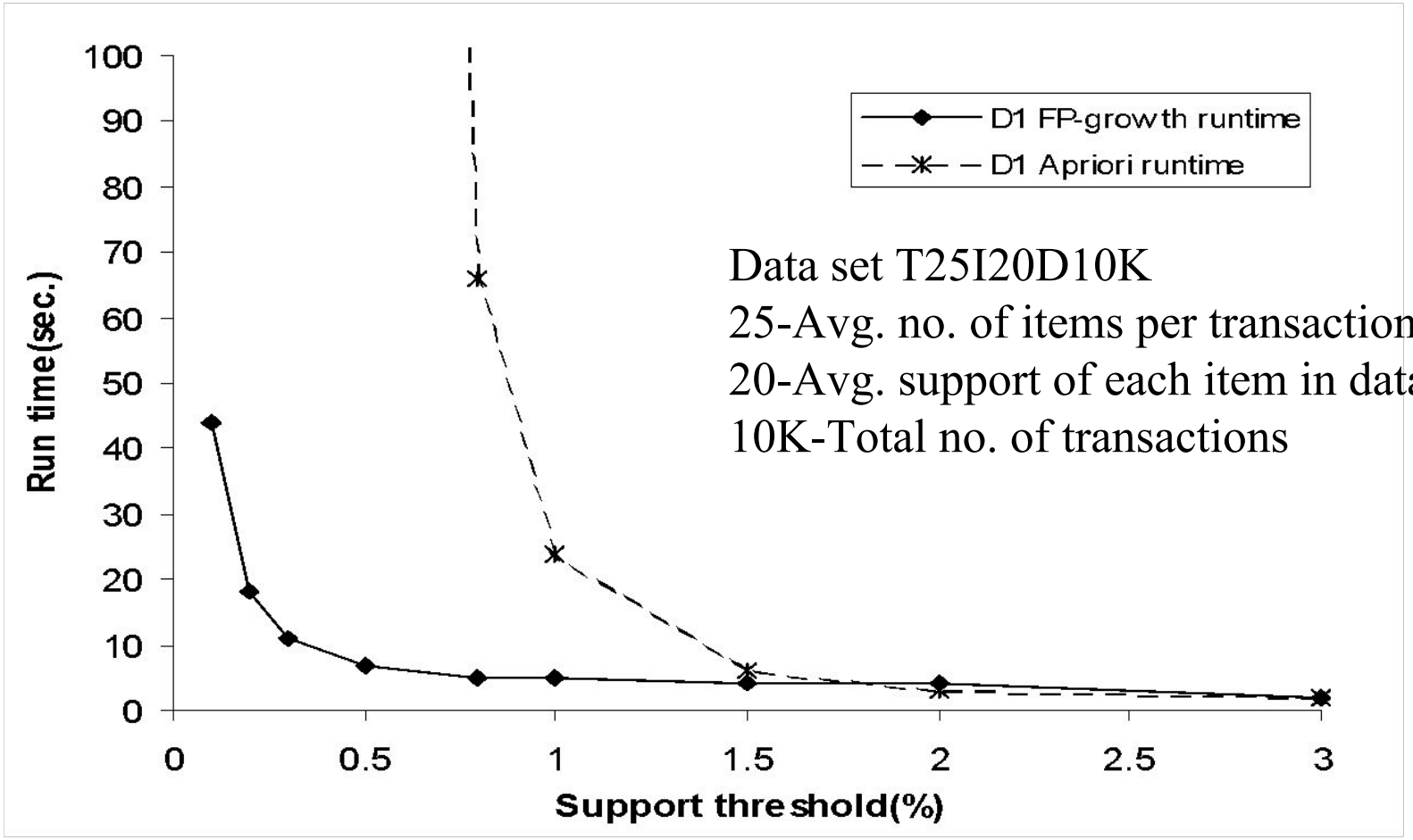
2. The FP-tree is mined by calling FP_growth(*FP_tree*, *null*), which is implemented as follows.

```
procedure FP_growth(Tree, α)
(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 β ∪ α 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 ∪ 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 ≠ ∅ then
(8)       call FP_growth(Tree_β, β); }
```

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

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



FP-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

- Performance is Faster than Apriori
  - Use compact data structure
  - No candidate generation, no candidate test
  - Eliminate repeated database scans
  - Basic operation is counting and FP-Tree building

- Problem:
  - When the database is large, sometimes unrealistic to construct a main memory based FP-Tree

Data Format

- Apriori and FP-Growth
  - { TID: itemset }
    - TID: Transaction ID
    - Itemset: set of items bought in transaction TID
  - Horizontal Data Format

- Alternative way
  - { Item: TID_set }
    - Item: item name
    - TID_set: set of transaction identifiers containing the item
  - Vertical Data Format

Data Format

Horizontal Data Layout

TID	Items
1	A,B,E
2	B,C,D
3	C,E
4	A,C,D
5	A,B,C,D
6	A,E
7	A,B
8	A,B,C
9	A,C,D
10	B

Vertical Data Layout

A	B	C	D	E
1	1	2	2	1
4	2	3	4	3
5	5	4	5	6
6	7	8	9	
7	8	9		
8	10			
9				

TID-list

Mining by Exploring Vertical Data Format

- ECLAT (Equivalence CLASS Transformation)
- Developed by Zaki

ECLAT Algorithm

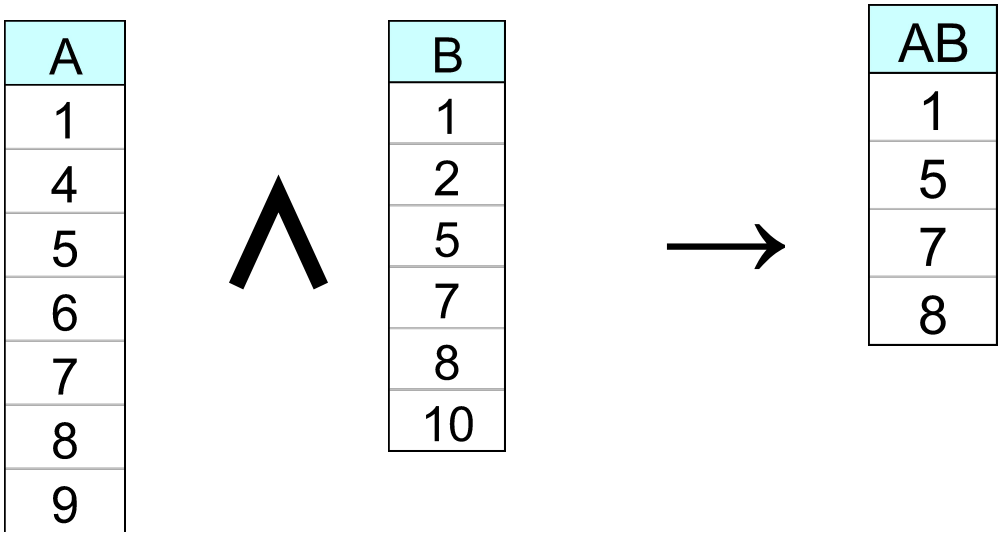
- Deriving frequent patterns based on vertical intersections
  - t(X) = t(Y): X and Y always happen together
  - t(X) ⊂ t(Y): transaction having X always has Y
- To count itemset AB
  - Intersect TID-list of itemA with TID-list of itemB

ECLAT Algorithm

- Transform the horizontally formatted data to the vertical format by scanning the data set once
- Support count of an itemset
  - The length of the TID_set of the itemset

ECLAT Algorithm

- Determine support of any k-itemset by intersecting tid-lists of two of its (k-1) subsets.



- 3 traversal approaches:
  - top-down, bottom-up and hybrid

ECLAT Algorithm

- Starting with k=1, the Frequent k-itemsets can be used to construct the candidate (k+1) itemsets based on the Apriori property
  - Done by intersection of the TID_sets of the frequent k-itemsets to compute the TID_sets of the corresponding (k+1) itemsets
- This process repeats, with k incremented by 1 each time, until no frequent itemsets or no candidate itemsets can be found

ECLAT Algorithm Summary

- Intersection is more efficient
- Pipelined counting for frequent itemsets
- Advantage
  - Less number of database scan
  - Very fast support counting
  - No need to scan the database to find the support of (k+1) itemsets (for k>=1)
    - Because the TID_set of each k-itemset carries the complete information required for counting each support

ECLAT Algorithm

- Disadvantage
  - Intermediate tid-lists may become too large for memory
  - Long computation time for intersecting the long set

- Performance improvement Idea
  - Using diffset to accelerate mining [CHARM Algorithm]
    - Only keep track of differences of tids
    - $t(X) = \{T_1, T_2, T_3\}$, $t(XY) = \{T_1, T_3\}$
    - Diffset (XY, X) = $\{T_2\}$

Problem of Frequent Item sets

- A long pattern contains a combinatorial number of sub-patterns
  - e.g., $\{a_1, \ldots, a_{100}\}$ contains

  $= \binom{100}{1} + \binom{100}{2} + \ldots + \binom{100}{100}$

  $= 2^{100} - 1$

  $= 1.27*10^{30}$ sub-patterns!
- Solution

Mine closed patterns and max-patterns instead

Closed Patterns

- An itemset X is closed if X is *frequent* and there exists *no super-pattern* Y ↄ X, *with the same support* as X
- It is a lossless compression of freq. patterns
  - Reducing the # of patterns and rules

Closed Patterns - Example

Transaction Database

1: $\{a, d, e\}$
2: $\{b, c, d\}$
3: $\{a, c, e\}$
4: $\{a, c, d, e\}$
5: $\{a, e\}$
6: $\{a, c, d\}$
7: $\{b, c\}$
8: $\{a, c, d, e\}$
9: $\{b, c, e\}$
10: $\{a, d, e\}$

Frequent Item Set

1 item	2 items	3 items
$\{a\}$: 7	$\{a, c\}$: 4	$\{a, c, d\}$: 3
$\{b\}$: 3	$\{a, d\}$: 5	$\{a, c, e\}$: 3
$\{c\}$: 7	$\{a, e\}$: 6	$\{a, d, e\}$: 4
$\{d\}$: 6	$\{b, c\}$: 3	
$\{e\}$: 7	$\{c, d\}$: 4	
	$\{c, e\}$: 4	
	$\{d, e\}$: 4	

- {b} is a subset of {b,c} both have a support of 3
- {d,e} is a subset of {a,d,e} both have a support of 4

All frequent item sets are Closed **except {b} and {d, e}**

Max-Patterns

- An itemset X is a max-pattern (maximal) if X is frequent and there exists no frequent super-pattern Y ⊃ X

Max-Patterns - Example

Transaction Database

1: $\{a, d, e\}$
2: $\{b, c, d\}$
3: $\{a, c, e\}$
4: $\{a, c, d, e\}$
5: $\{a, e\}$
6: $\{a, c, d\}$
7: $\{b, c\}$
8: $\{a, c, d, e\}$
9: $\{b, c, e\}$
10: $\{a, d, e\}$

Frequent Item Set

1 item	2 items	3 items
$\{a\}$: 7	$\{a, c\}$: 4	$\{a, c, d\}$: 3
$\{b\}$: 3	$\{a, d\}$: 5	$\{a, c, e\}$: 3
$\{c\}$: 7	$\{a, e\}$: 6	$\{a, d, e\}$: 4
$\{d\}$: 6	$\{b, c\}$: 3	
$\{e\}$: 7	$\{c, d\}$: 4	
	$\{c, e\}$: 4	
	$\{d, e\}$: 4	

The maximal item sets are {b,c} {a,c,d} {a,c,e} {a,d,e}

- Every frequent itemset is a subset of at least one of these sets

Closed Patterns and Max-Patterns

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

Mine the Closed and Max-Patterns

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

Minimum support = 2