

Data Mining:

Mining Association Rules



What is Association Rule Mining?

- Association rules mining
 - Discovering **interesting** relations between **objects** in large databases
 - Frequent patterns, associations, correlations, or causal structures among sets of items or objects
 - Transactional databases, relational databases, and other information repositories.
 - Different measures of interestingness
- Market basket analysis
 - The problem is to analyse customer buying habits by finding **associations between the different items** that customers place in their “shopping baskets”

A Motivating Example: Market Basket Analysis



Anything interesting?

Bread → Milk (100%)
Diapers → Beer (66%)
Diapers → Milk (100%)

Customers who buy diapers also tend to buy beer



Identify **potential cross-selling opportunities** among related items



Motivation (market basket analysis)

- If customers are buying milk, how likely is that they also buy bread?
- Such rules help retailers to:
 - Plan the shelf space: by placing milk close to bread they may increase the sales
 - Provide advertisements/recommendation to customers that are likely to buy some products
 - Put items that are likely to be bought together on discount, in order to increase the sale



Problem Statement

- Given:
 - database of transactions
 - Each transaction is a list of items (purchased by a customer in a visit)
- Find:
 - All rules that correlate the presence of one set of items with that another set of items
 - E.g., 80% of customers who buy {diapers} tend to buy {beer, milk} as well.

Transaction data: supermarket data

- Market basket transactions:

t1: {bread, cheese, milk}

t2: {apple, eggs, salt, yogurt}

... ..

tn: {biscuit, eggs, milk}

- Concepts:

- *An item*: an item 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

- A 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

Example of a Transaction Database

This is a sample input of the algorithm to be discussed.

Transaction Database	Items
100	1 2 3 4 5
200	6 2 3 5
300	1 2 4
400	2 9 4 5
500	1 2 6 4
600	6 3 5
...	...



Formal Notations

- A set of items $\mathcal{I} = \{i_1, i_2, \dots, i_n\}$
- A transaction database $\mathcal{D} = \{t_1, t_2, \dots, t_m\}$
 $t_i \subseteq \mathcal{I}$, (a transaction is a set of items)
TID: $t_i \rightarrow$ transaction number
- A transaction t contains X , a set of items (itemset) in \mathcal{I} , if $X \subseteq t$.
 - An **itemset** is a set of items.
 - E.g., $X = \{\text{milk, bread, cereal}\}$ is an itemset.
 - A **k -itemset** is an itemset with k items.
 - E.g., $\{\text{milk, bread, cereal}\}$ is a 3-itemset



Formal Notations (cont.)

- An **association rule** is an implication of the form:

$X \rightarrow Y$, where $X, Y \subset I$, and $X \cap Y = \emptyset$
($X \Rightarrow Y$)

e.g.: Bread, Butter \rightarrow Milk

Buy(Bread, Butter) \rightarrow Buy(Milk)



Mining Association Rules

- Boolean Association Rule

- Concerns the associations between presence or absence of items.

Computer \Rightarrow financial_management_software

- Quantitative Association Rule

- Describes the associations between quantitative items.
- Discretized quantitative values for items or attributes are partitioned into intervals.

Age(X, "30 .. 39") \wedge income(X, "42K .. 48K") \Rightarrow buys(X, high resolution TV)



Mining Association Rules (cont.)

- Single Dimensional Association Rule
 - Only one dimension (attribute) is involved
buys(Computer) ⇒ buys(financial_management_software)
- Multi-Dimensional Association Rule
 - Two or more dimensions (attributes) are involved in the rule.
age(X, "30 .. 39") ∧ income(X, "42K .. 48K") ⇒ buys(X, high resolution TV)
- Single or Multi-level association Rule
 - based on the levels of abstractions involved in the rule set
age(X, "30 .. 39") ⇒ buys(X, "laptop computer")
age(X, "30 .. 39") ⇒ buys(X, "computer")

Mining on multidimensional association rules is not discussed in this course.



Support and Confidence

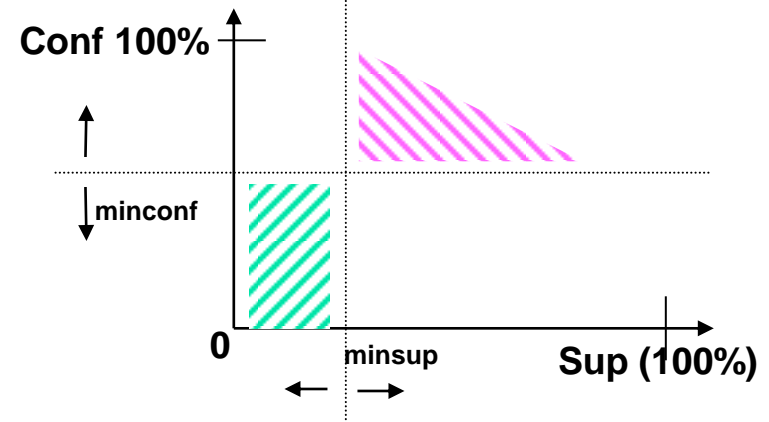
- Rule $X \Rightarrow Y$ or $X \rightarrow Y$
- Rule support
 - (absolute) support or support count: **frequency** or **occurrence** of an itemset $X \cup Y$
 - (relative) support: **probability** that a transaction contains $X \cup Y$
support $(X \Rightarrow Y) = P(X \cup Y)$
 $s\%$ of transactions in \mathcal{D} contain $X \cup Y$ (i.e., both X and Y)
- Rule confidence:
 - conditional probability

P for probability.

Confidence $(X \Rightarrow Y) = P(Y | X) = P(X \cup Y) / P(X)$
transactions in \mathcal{D} contain X has $c\%$ containing Y

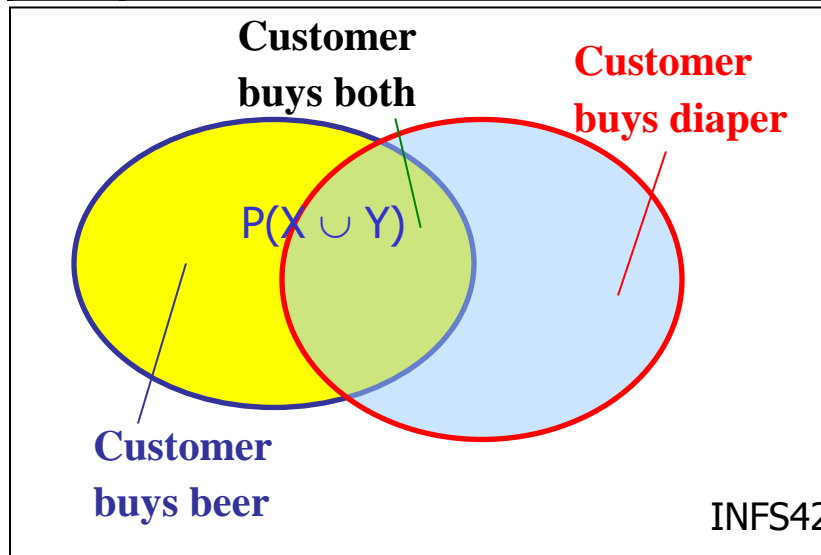
Support and Confidence (cont.)

- User-specified threshold:
 - *min_sup*
 - *min_conf*
 - An itemset X is *frequent* if X's support is no less than *min_sup*
 - Rules that satisfy *min_sup* and *min_conf* are called **strong**



Example

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	support count	support
Milk	2	40%
Beer		
Nuts		
Diaper		
Eggs		
Beer, Diaper		
...		

min_sup=50%

Example

Find all the rules $X \rightarrow Y$ with minimum support and confidence
support, s , **probability** that a transaction contains $X \cup Y$
confidence, c , **conditional probability** that a transaction having X also contains Y

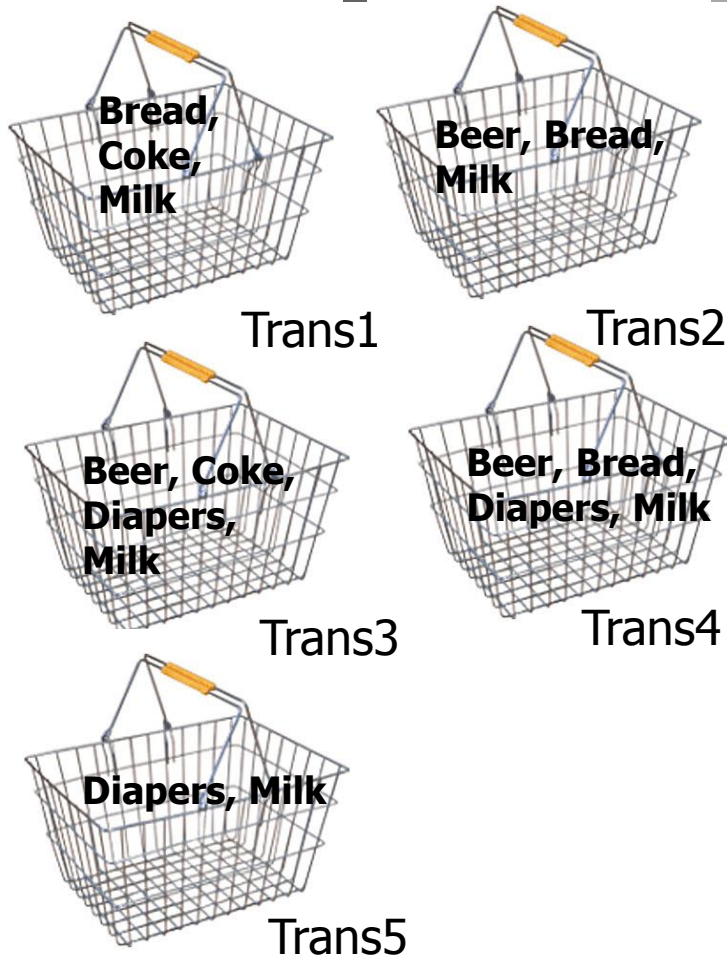
Tid	Items bought
10	Beer, Nuts, Diaper
20	Beer, Coffee, Diaper
30	Beer, Diaper, Eggs
40	Nuts, Eggs, Milk
50	Nuts, Coffee, Diaper, Eggs, Milk

min_sup=50%
min_conf = 50%

Rule: Diaper \rightarrow Beer
support: 3 (60%)
confidence: $P(\text{Beer, Diaper})/P(\text{Diaper})$
 $= 60\%/80\% = 75\%$

Rule: Beer \rightarrow Diaper
support: 3 (60%)
confidence: $P(\text{Beer, Diaper})/P(\text{Beer})$
 $= 60\%/60\% = 100\%$

An Example



Bread \Rightarrow Milk
Diapers \Rightarrow Beer
Diapers \Rightarrow Milk

Support_count(Bread \Rightarrow Milk) = ?
Confidence (Bread \Rightarrow Milk) = ?

Support_count(Diapers \Rightarrow Beer) = ?
Confidence (Diapers \Rightarrow Beer) = ?



The Problem Statement

- Find all sets of items (itemsets) that have transactions support above *min_sup* (i.e., find all **large** itemsets).
- Use **large** itemsets to generate association rules.

In this discussion, “large” means “frequent”:

$\text{count} \geq \text{min_sup}$



Mining Association Rules

- Apriori Algorithm
- Frequent Pattern (FTP) Tree Algorithm



Apriori Algorithm

- Finding frequent itemsets using candidate generation
 - $C_1 \rightarrow L_1 \rightarrow \dots \rightarrow C_k \rightarrow L_k \rightarrow C_{k+1} \rightarrow L_{k+1} \rightarrow \dots$
- Set of candidate k-itemsets
 - C_k (itemset)
 - E.g., $C_2 = \{\{\text{Bread, Milk}\}, \{\text{Diaper, Beer}\}, \{\text{Diaper, Milk}\}\}$
- Large itemset (frequent itemset):
 - L_k (itemset, support_count)
 - support_count > min_sup
 - E.g., $L_2 = \{\{\text{Bread, Milk}\}, \{\text{Diaper, Milk}\}\}$, where min_sup=3

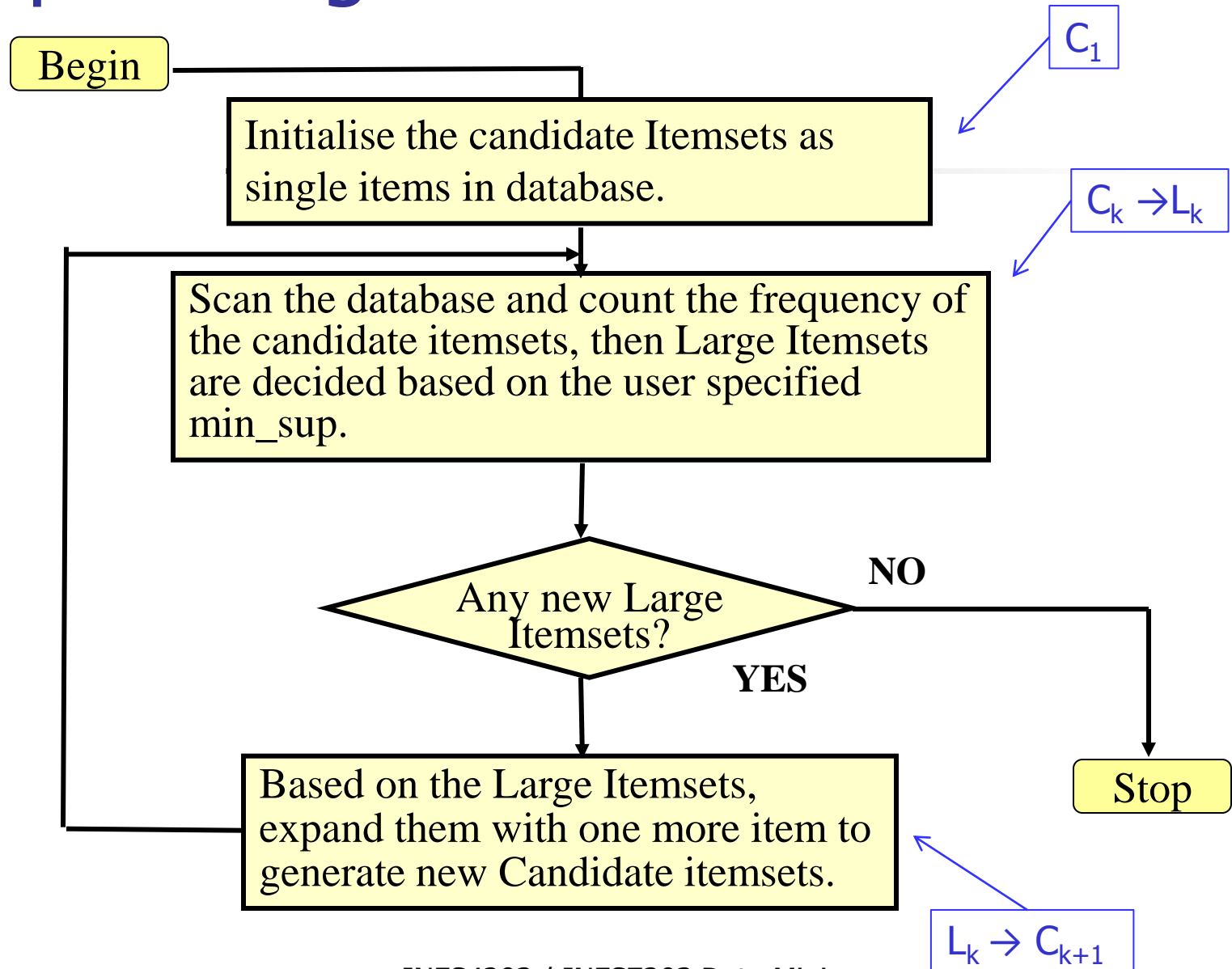


What does it mean by “Apriori”?

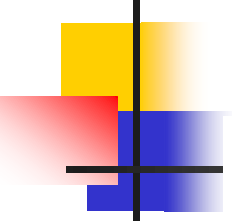
- Apriori Property:
 - All nonempty subsets of a large (frequent) itemset must also be large (frequent), or
 - If an itemset is not large (frequent), all its supersets cannot be large (frequent). This is called **anti-monotone**.
 - This property is used to create the candidate k -itemsets C_k based on L_{k-1} , then test for L_k .

This is a discovered truth not an assumption.

Apriori Algorithm



Apriori Algorithm

- 
- 1) $L_1 = \{\text{large 1-itemsets}\}$
 - 2) for ($k=2; L_{k-1} \neq \emptyset; k++$) do begin
 - 3) $C_k = \text{apriori-gen}(L_{k-1});$ // New candidates
 - 4) for each transactions $t \in \mathcal{D}$ do begin
 - 5) $C_t = \text{subset}(C_k, t)$ // get the subsets of t that are candidates
 - 6) for each candidate $c \in C_t$ do
 - 7) $c.\text{count} ++;$
 - 8) end
 - 9) $L_k = \{c \in C_k \mid c.\text{count} \geq \text{min_sup}\}$
 - 10) end
 - 11) Answer = $\bigcup_k L_k;$

Apriori-gen Procedure

Input: L_{k-1}

Return: C_k

// **Generate the candidate itemsets C_k from L_{k-1}**

Step 1: Join L_{k-1} with L_{k-1} to create new candidates

SELECT INTO C_k

SELECT $p.item_1, p.item_2, \dots, p.item_{k-1}, q.item_{k-1}$

FROM $L_{k-1} p, L_{k-1} q$

WHERE $p.item_1 = q.item_2, \dots, p.item_{k-2} = q.item_{k-2},$
 $p.item_{k-1} < q.item_{k-1};$

Step 2: Prune: Delete itemsets $c \in C_k$ if some $(k-1)$ subsets of c is not in L_{k-1}

for each itemsets $c \in C_k$ do

for each $k-1_subsets$ s of c do

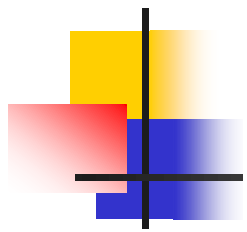
if ($s \notin L_{k-1}$) then

delete c from C_k ;

end;

**How to justify
this deletion?**

Apriori Example



Database

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

C₁

Itemset	Support Count
{1}	2
{2}	3
{3}	3
{4}	1
{5}	3

L₁

Itemset	Support
{1}	2
{2}	3
{3}	3
{5}	3




C₂

Itemset	Support
{1 2}	1
{1 3}	2
{1 5}	1
{2 3}	2
{2 5}	3
{3 5}	2

Min_Sup = 2

Does {1 4} need
be generated?

Apriori Example



L₂

Itemset	Support
{1 3}	2
{2 3}	2
{2 5}	3
{3 5}	2

Min_Sup = 2



C₃

Itemset	Support
{2 3 5}	2

L₃

Itemset	Support
{2 3 5}	2

Does {1 2 3} need
be generated?

Note that: {1,2,3}
will **not** be in C₃
since one of its
subsets {1,2} is not
in L₂.

Problem in this Algorithm:

When Database is scanned to check C_k for creating L_k, a large number of transactions will be scanned even they do not contain any k-itemset.

Maximum Frequent Itemsets: Max-patterns

- Frequent pattern $\{a_1, \dots, a_{100}\} \rightarrow \sum_{k=1}^{100} \binom{100}{k} = 2^{100}-1 = 1.27 \times 10^{30}$ frequent sub-patterns!
- Max-pattern: frequent patterns without proper frequent super pattern
 - BCDE, ACD are max-patterns
 - BCD is not a max-pattern

min_sup=2

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

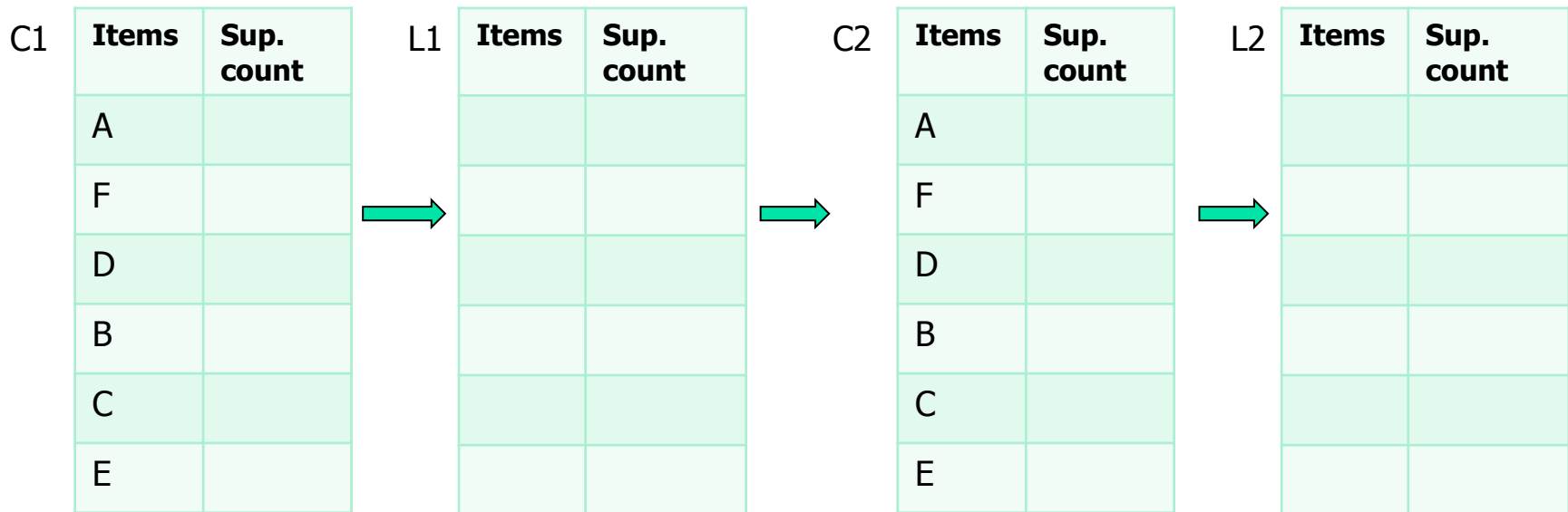


How to Generate Association Rules from Frequent Itemsets?

- Create Rules for Maximum Frequent Itemsets:
No trivial rules.
- Confidence $(A \Rightarrow B) = P(B|A) = \text{Support_Count}(A \cup B) / \text{Support_Count}(A)$
 - For each frequent itemset L , generate all non-empty subsets of L .
 - For every nonempty subset S of L , generate rule: $S \Rightarrow (L - S)$, then calculate its confidence
 $c = \text{support_count}(L) / \text{Support_Count}(S)$.
If $c \geq \text{min_conf}$, the rule is a strong association rule.
 - e.g., $L = A, B, C$; rules: $A \Rightarrow B, A \Rightarrow C, A \Rightarrow BC, AB \Rightarrow C...$

TID	Date	Items Bought
1	5/10/02	{ <u>A</u> pple, <u>F</u> ish, <u>D</u> etergent, <u>B</u> anana}
2	5/10/02	{ <u>D</u> etergent, <u>B</u> anana, <u>A</u> pple, <u>C</u> arrot, <u>E</u> gg }
3	10/10/02	{ <u>C</u> arrot, <u>B</u> anana, <u>A</u> pple, <u>E</u> gg}
4	20/10/02	{ <u>B</u> anana, <u>D</u> etergent, <u>A</u> pple }

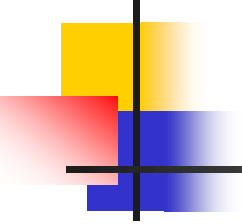
Min_sup: 60%
Min_conf: 80%





Problems with Apriori Algorithms

- It is costly to handle a huge number of candidate sets. For example if there are 10^4 large 1-itemsets, the Apriori algorithm will need to generate more than 10^7 candidate 2-itemsets. Moreover for 100-itemsets, it must generate more than $2^{100} \approx 10^{30}$ candidates in total.
- The candidate generation is the inherent cost of the Apriori Algorithms, no matter what implementation technique is applied.

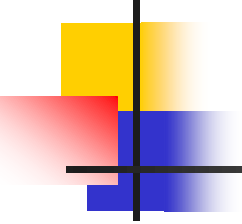


It is **not** a good idea to use
Apriori Algorithms to mine
a large database for long
patterns.



Mining Frequent Patterns Without Candidate Generation

- 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



Frequent Pattern Tree Algorithm

("FP-Growth Algorithm")

1

- Items in transactions are sorted based on their **frequencies** in the database (in **descending order**).
- A tree structure (Prefix-Tree) is used to record the frequent patterns top down. Only **frequent (large) item will have nodes** in the tree (i.e., database is compressed).

2

- Each projection on the tree is to mine all frequent patterns associated with the item (in the Header Table) and the procedure is **recursive**.
- It is not Apriori like restricted generate-and-test but restricted test (divide-and-conquer) only.

Construct FP-tree from a Transaction Database

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}

TID	Ordered frequent items
100	{f, c, a, m, p}
200	{f, c, a, b, m}
300	{f, b}
400	{c, b, p}
500	{f, c, a, m, p}

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

support count:

f	c	a	b	m	p	o
4	4	3	3	3	3	2

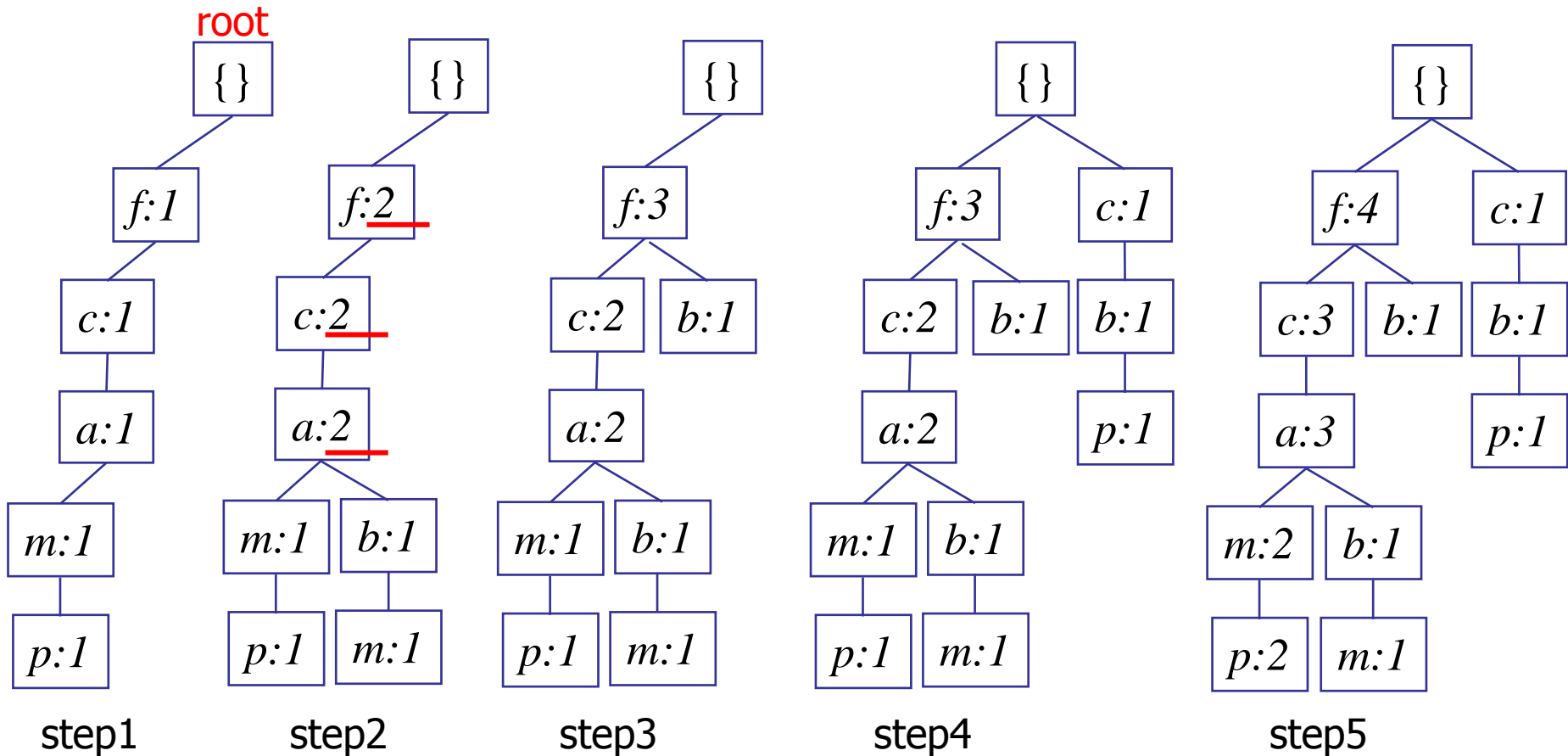
min_sup = 3

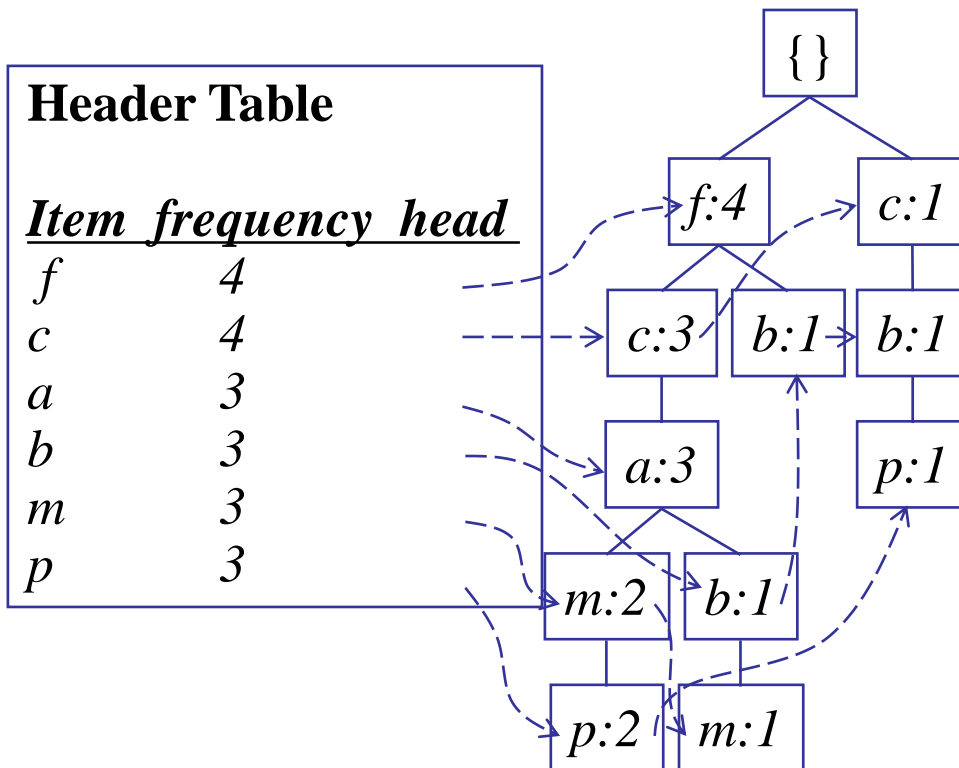
step 2: Sort frequent items in frequency descending order

step 3: Scan DB again, construct FP-tree

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

When considering the branch to be added for a transaction, the count of each node along a common prefix is incremented by 1





An **item header** table is used to link all leaf nodes in a list.

FP Growth Algorithm (part 1)

Building the FP-Tree:

Initialise the Frequent (large) Itemsets as single items in database. Sort the **1-itemsets** in the descending order.

Create the root of FP-Tree and label it as “null”.

Construct the tree by scanning the transactions in the database according to the order of the **1-itemsets**. Create a branch in the tree if there is no common prefix in the path of the tree. The counting is performed for the items in the transaction along the path of the tree.

An **item header** table is used to link all leaf nodes in a list.

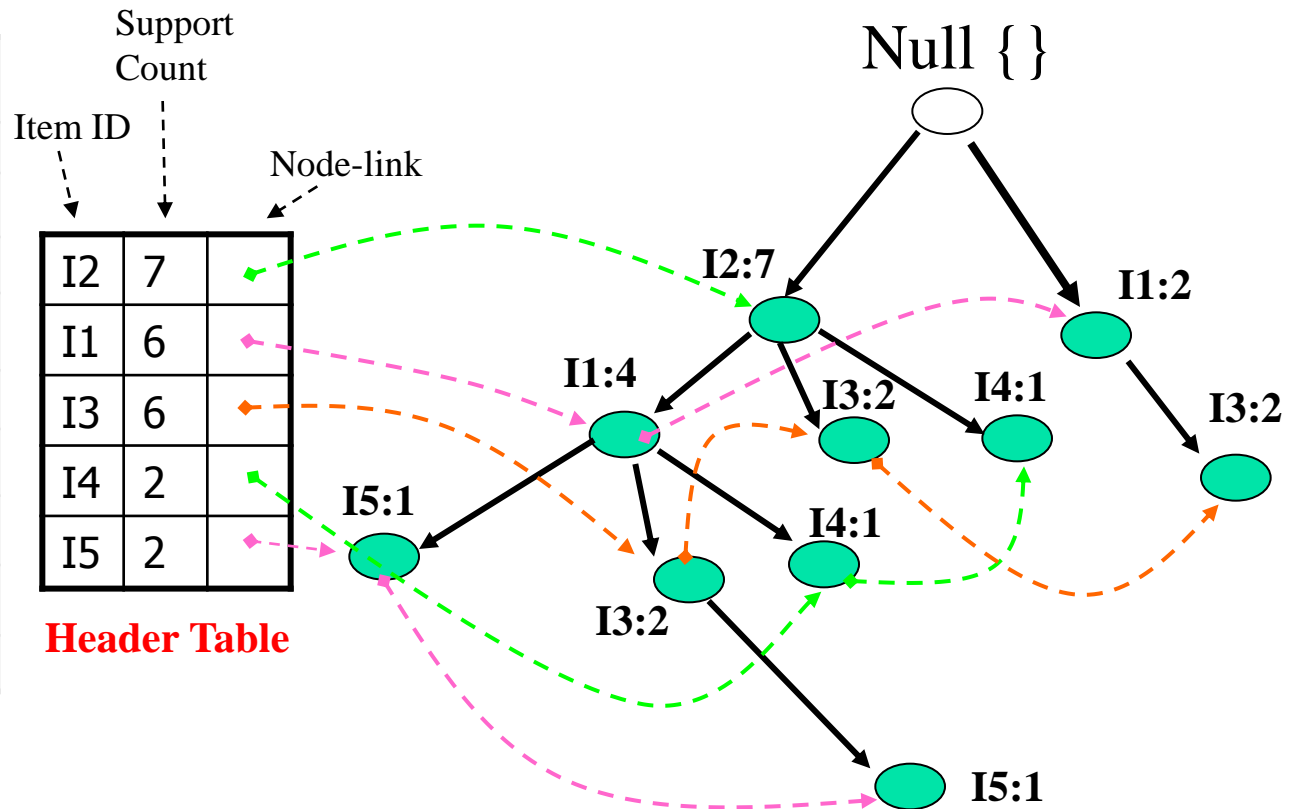
How many scans of the database in order to build the FP-Tree?

Mine frequent patterns from FP-tree

FP-Tree Construction:

Database

TID	List of item_IDs	Ordered by frequency
100	I1, I2, I5	I2, I1, I5
200	I2, I4	I2, I4
300	I2, I3	I2, I3
400	I1, I2, I4	I2, I1, I4
500	I1, I3	I1, I3
600	I2, I3	I2, I3
700	I1, I3	I1, I3
800	I1, I2, I3, I5	I2, I1, I3, I5
900	I1, I2, I3	I2, I1, I3



The FP-Tree with
min_sup = 2

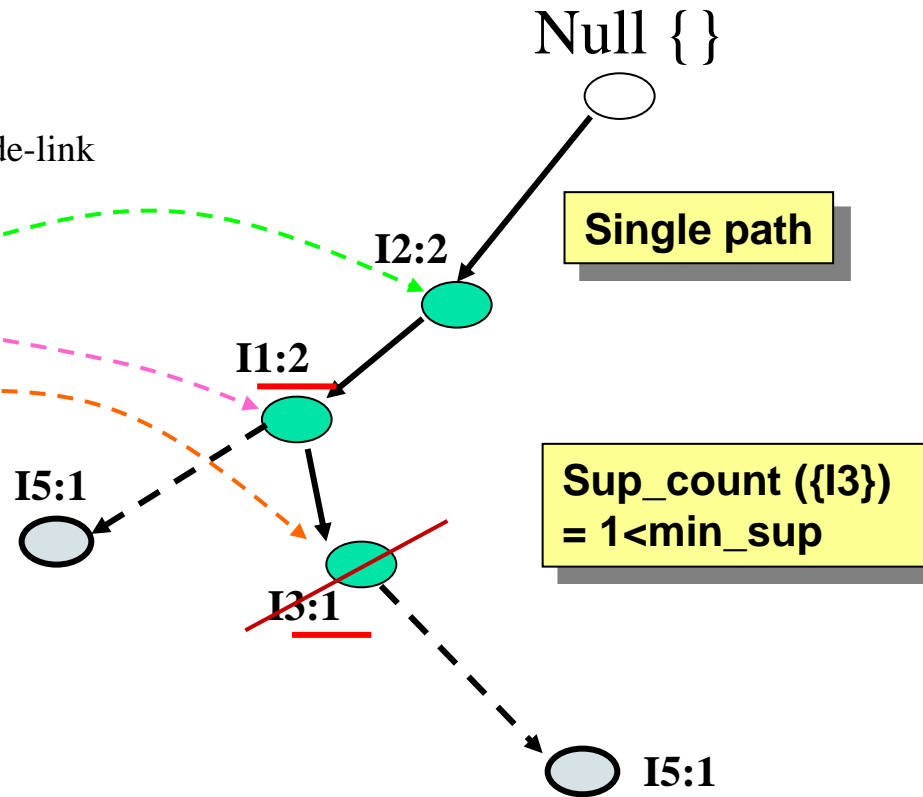
The **conditional FP-Tree** associated with the conditional node I5.

Step 1: Start from the last item header table as the suffix pattern, construct its **conditional pattern base**.

min_sup = 2

I2	2	→
I1	2	→
I3	1	→

Header Table



Step 2:

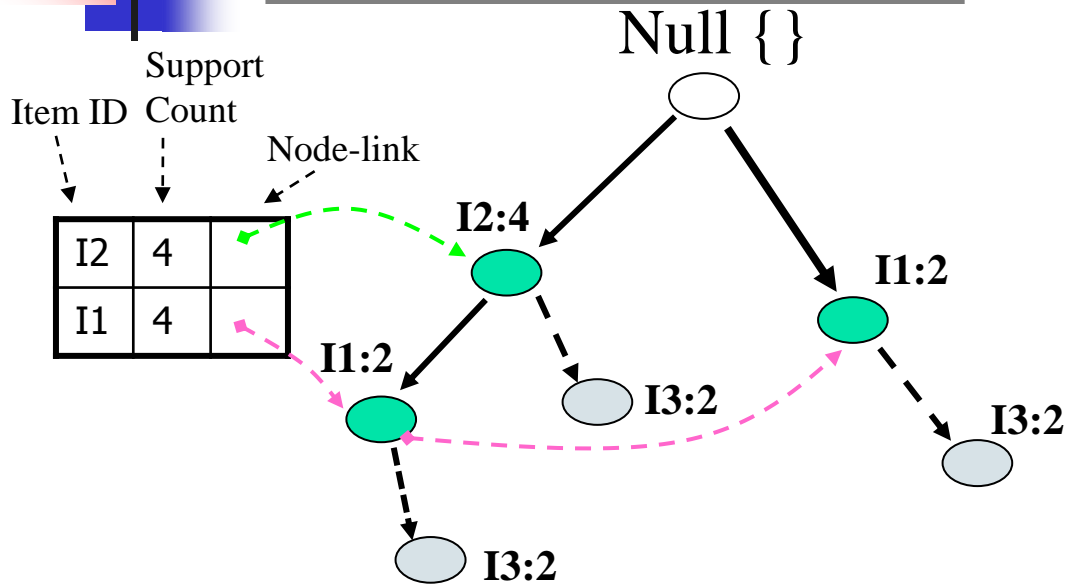
Conditional pattern base (**prefix path**) for I5:
 { (I2 I1: 1), (I2 I1 I3: 1) }

Frequent pattern for I5:

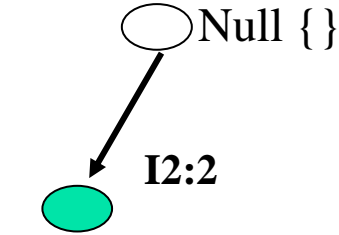
I2 I5: 2	I2 I1 - I5 → I2 I5 :2
I1 I5: 2	I1 I5 :2
I2 I1 I5: 2	I2 I1 I5 :2

Along the path from the prefix, all the counting (support) will be the same number as the suffix.

The conditional FP-Tree associated with the conditional node I3.

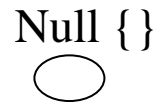


I2 I3: 4, I1 I3: 4, I2 I1 I3: 2



The conditional FP-Tree with node I1.

I2 I1: 2, I1: 2



The conditional FP-Tree with node I2.

I2: 2

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

Another example:

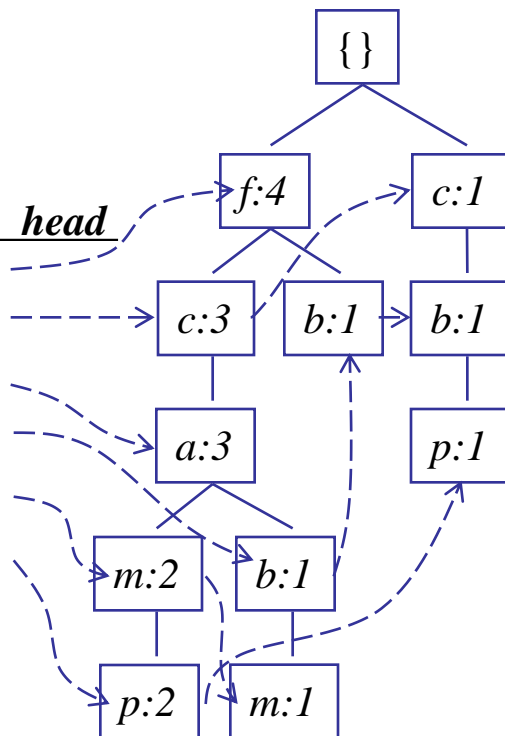
Find Patterns Having P From P-conditional Database

- 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.

Header Table

Item frequency head

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



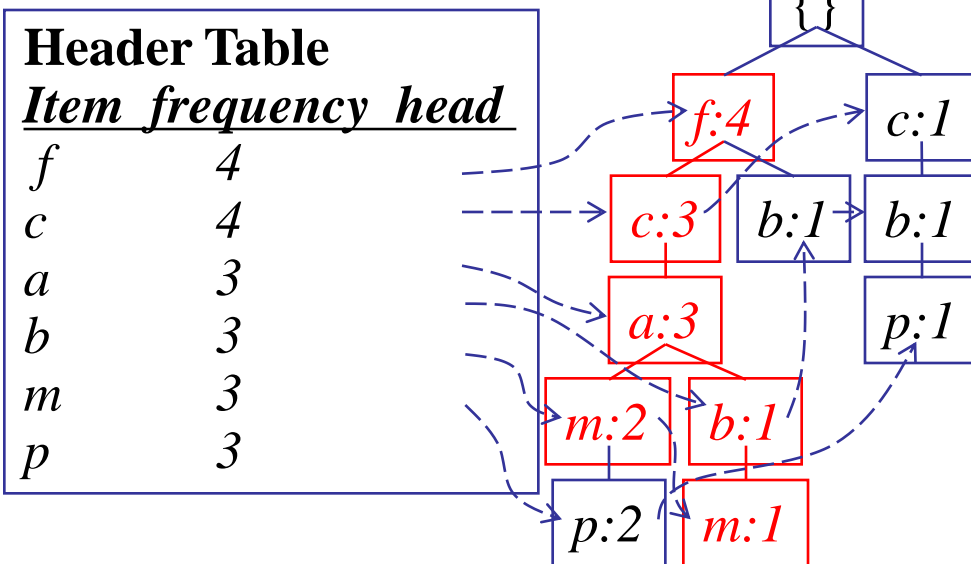
Conditional pattern bases

item cond. pattern base

c	$f:3, \{\}$
a	$fc:3$
b	$fca:1, f:1, c:1$
m	$fca:2, fcab:1$
p	$fcam:2, cb:1$

Example (cont.)

- For each pattern-base
 - Accumulate the count for each item in the base
 - Construct the FP-tree for the frequent items of the pattern base



m-conditional pattern base:

fca:2, fcab:1

{}

f:3

c:3

a:3



All frequent patterns relate to *m*

m,

fm, cm, am,

fcm, fam, cam,

fcam

m-conditional FP-tree

Recursion: Mining Each Conditional FP-tree

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

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

↓
c:3

↓
a:3

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

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

{ }

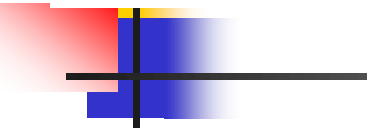
↓
f:3

Stop at { }

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

FP Growth Algorithm (part 2)

Mining Frequent Patterns from the FP-Tree:



Start from the last item of header table as the suffix pattern, construct its **conditional pattern base**. Along the path from the prefix, all the counting (support) will be the same number as the suffix.

Based on the conditional pattern base, construct the item's **conditional FP-Tree**, and performing 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**.

The algorithm stops when the traversal of the header table finishes. All the **frequent patterns** (i.e., large itemsets) are created in the process of the pattern growth.

Summary:

Mining Frequent Patterns With FP-trees

- Idea: Frequent pattern growth
 - Recursively grow frequent patterns by pattern and database partition
- Method
 - For each frequent item, construct its **conditional pattern-base**, and then its **conditional FP-tree**
 - Repeat the process on each newly created conditional FP-tree
 - Until the resulting FP-tree is empty, or it contains only one path—single path will generate all the combinations of its sub-paths, each of which is a frequent pattern



Benefits of the FP-tree Structure

- **Completeness**
 - Preserve complete information for frequent pattern mining
 - Never break a long pattern of any transaction
- **Compactness**
 - Reduce irrelevant info—infrequent items are gone
 - Items in frequency descending order: the more frequently occurring, the more likely to be shared
 - Never be larger than the original database (not count node-links and the *count* field)

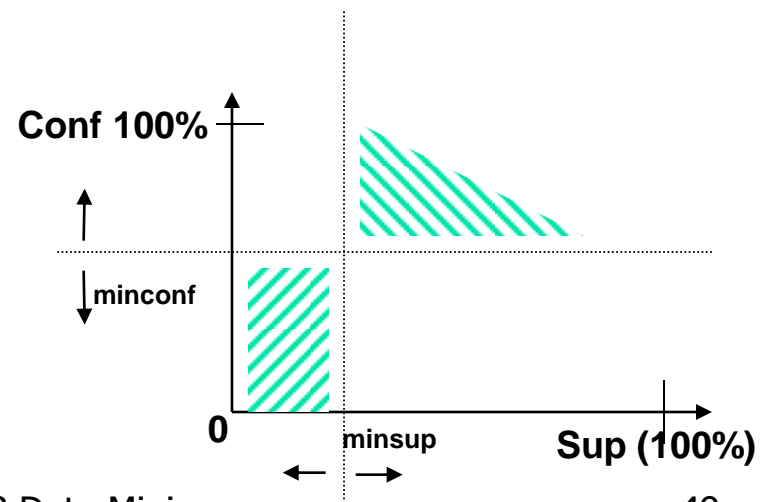


Why Is FP-Growth the Winner?

- Divide-and-conquer:
 - decompose both the mining task and DB according to the frequent patterns obtained so far
 - leads to focused search of smaller databases
- Other factors
 - no candidate generation, no candidate test
 - compressed database: FP-tree structure
 - no repeated scan of entire database
 - basic ops—counting local freq items and building sub FP-tree, no pattern search and matching

Current Research Issues in Association Rules Mining

- Explanation and Interpretations
 - Causal Relationship/Dependence discovery
- Automatically determining the minimum support and confidence
- High-dimensionality verses Hierarchical mining
- Mining on streaming data
- Mining Negative Rules
 - $A \Rightarrow \neg B, \neg A \Rightarrow B, \neg A \Rightarrow \neg B$





Reading List: Association Rules

- Rakesh Agrawal and Ramakrishnan Srikant, [*Fast Algorithms for Mining Association Rules*](#), Proc of 20th VLDB Conference Santiago, Chile, 1994.*
- Jiawei Han, Jian Pei, and Yiwen Yin, [*Mining Frequent Patterns without Candidate Generation*](#) In Proce. Of the 2000 ACM-SIGMOD int'l Conf. On Management of Data, Dallas, Texas, USA, May 2000.*
- Jochen Hipp, Ulrich Guntzer, and Gholamreza, [*Algorithms for Association Rule Mining – A general Survey and Comparison*](#) ACM SIGKDD Explorations, Vol 2 Issue 1, July 2000.