SDSC3002
Market Basket Analysis: Frequent Pattern Mining

Yu Yang
yuyang@cityu.edu.hk

Outline

What is Frequent Pattern Mining?

- Data Mining: extracting patterns from massive data
  - Pattern: a set of items, subsequences, or substructures that occur together frequently in a data set
- Motivation: uncovering inherent regularities in data

Frequently bought together

Total price: CDN$ 42.90

Add both to Cart

☑ **This item:** Huggies Natural Care Fragrance-Free Baby Wipes, Refill Pack, 1056 Count CDN$ 19.97 (CDN$ 0.02 / count)

☑ Playtex Diaper Genie Diaper Pail System Refills, 3 pack CDN$ 22.93 (CDN$ 7.64 / ring)

What is Frequent Pattern Mining?

- Data Mining: extracting patterns from massive data
  - Pattern: a set of items, subsequences, or substructures that occur together frequently in a data set
- Motivation: uncovering inherent regularities in data

```
void __init prom_meminit(void)
{
    ......
    for (i=0; i<n; i++) {
        total[i].adr = list[i].addr;
        total[i].bytes = list[i].size;
        total[i].more = &total[i+1];
    }
    ......
for (i=0; i<n; i++) {
        taken[i].adr = list[i].addr;
        taken[i].bytes = list[i].size;
        taken[i].more = &total[i+1];
    }
```

Common Patterns in Code: likely specifications and properties

Violation of Patterns: maybe bugs

Mining **Sequetial Patterns** to **Detect Copy-and-Paste Bugs**

Why Frequent Patterns are Important?

- **Fruitful applications**
  - Basket data analysis, cross-marketing, catalog design, sale campaign analysis, web log (click stream) analysis, ...
- **Fundamental step** of many data mining tasks
  - Association, correlation, causality analysis
  - Time-series analysis (sequential patterns)
  - Graph mining, Graph similarity/kernel (sub-graph patterns)
  - Classification (discriminative patterns)
  - ...

Frequent Patterns in Transaction/Set Data

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

- Pattern/Itemset: a set of items
  - The most fundamental type of patterns
- k-itemset $I = \{i_1, ..., i_k\}$: a set of k items
  - $\{Beer, Diaper\}$ is a 2-itemset
- Support of I: #transactions containing all items in I
  - Frequency/Relative Support: $Freq(I) = \frac{Sup(I)}{|TBD|}$
  - $I = \{Beer, Diaper\}$, $Sup(I) = 3$, $Freq(I) = 0.6$

Frequent Patterns in Transaction/Set Data

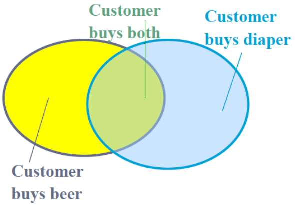
Tid	Items bought
10	Beer, Nuts, Diaper
20	Beer, Coffee, Diaper
30	Beer, Diaper, Eggs
40	Nuts, Eggs, Milk
50	Nuts, Coffee, Diaper, Eggs, Milk

- Minimum support *min_sup*
  - Defined by users of FPM
- I is a frequent itemset if $Sup(I) \geq min\_sup$
  - $min\_sup = 3$, {Beer, Diaper} is frequent, {Nuts, Diaper} is not
- Frequent Pattern/Itemset Mining
  - Input: a transaction DB, *min_sup*
  - Output: all frequent itemsets

Outline

Market-Basket Analysis: Association Rules

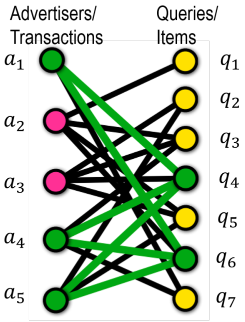
Tid	Items bought
10	Beer, Nuts, Diaper
20	Beer, Coffee, Diaper
30	Beer, Diaper, Eggs
40	Nuts, Eggs, Milk
50	Nuts, Coffee, Diaper, Eggs, Milk



- If one buys diapers, what else is she likely to buy?
- Association rule $X \rightarrow Y$
  - Confidence: $Conf(X \rightarrow Y) = \frac{Sup(X \cup Y)}{Sup(X)} \geq min\_conf$
  - Support: $Sup(X \cup Y) \geq min\_sup$ (**the harder part**)
- $min\_conf = 50\%$, $min\_sup = 3$
  - Frequent itemsets: ({Beer},3), ({Nuts},3), ({Diaper},4), ({Egg},3), ({Beer,Diaper},3)
  - Association Rules: {Beer}→{Diaper} (100%,3), {Diaper}→{Beer} (75%,3)

Sub-Market Extraction: Bipartite Clique Analysis

- Sponsored Search: displaying ads when relevant queries are issued
- Bipartite graph $G = (A \cup Q, E)$
  - A is the set of advertisers
  - Q is the set of queries
  - (a, q) is an edge if the advertiser a is willing to spend money on the query q
- (n, m) Sub-Market
  - n advertisers and m queries that are fully connected
  - An (n, m) sub-market is a frequent m-itemset with $min\_sup = n$



$\{a_1, a_4, a_5\} \cup \{q_4, q_6\}$ is a **sub-market**
a_1, a_4, a_5 are **competitors**
q_4 and q_6 may **summarize important features** of products of a_1, a_4, a_5

Outline

Outline

Challenges

- Challenge 1: #candidate_Itemsets
  - A naive idea: generate all possible itemsets and test supports
  - **Assume** we have 200 items: $2^{200} - 1 \approx 1.6 \times 10^{60}$ candidates
  - Age of universe$\approx 4.3 \times 10^{17} s$, IBM Summit: $2 \times 10^{17} Flops$, $4.3 \times 10^{17} s \times 2 \times 10^{17} \ll 1.6 \times 10^{60}$
  - **Reality**: Amazon.com has more than 17,000 books (items) relevant to data mining
- Challenge 2: counting supports of a huge number of itemsets
  - Walmart has more than 20 million transactions per day
  - I/O is costly
- **Efficiency is a real demand!**

How to Get an Efficient Method?

Building Block 1

Reducing #candidate_itemsets that need to be checked

Building Block 2

Counting supports of itemsets efficiently

Outline

Anti-Monotonicity of Itemsets

- Given two itemsets I_1 and I_2, if $I_1 \subseteq I_2$
  - Any transaction containing I_2 must contains I_1
  - A transaction containing {beer, diaper, nuts} also contains {beer, diaper}
  - $Sup(I_2) \leq Sup(I_1)$
- Any superset of an infrequent itemset must also be infrequent
  - {beer,diaper} is infrequent $\Rightarrow$ {beer,diaper,nuts} is infrequent
  - No superset of infrequent itemset should be generated
  - **Many item combinations can be pruned!**

How Apriori Works

- Level-wise, candidate generation and test
  - First mine frequent 1-itemsets, then frequent 2-itemsets, ...
- $L_1 = \{frequent\ items\}$, scan TDB once to compute
- Level $k \geq 2$
  - C_k={candidate itemsets of size k}
  - L_k={frequent itemsets of size k}
- Stops if $L_i = \emptyset$ at a level i

Pseudo Code of Apriori

Algorithm 1 Apriori

Input: a transaction DB and *min_sup*
Output: all frequent itemsets

$L_1 \leftarrow \{\textit{frequent items}\}$
for $k = 2$; $L_{k-1} \neq \emptyset$; $k \leftarrow k + 1$ **do**
    $C_k \leftarrow$ candidates generated based on L_{k-1} **(Candidate Generation)**
    Scan Transaction DB to count supports of itemsets in C_k **(Counting Supports)**
    $L_k \leftarrow$ candidates in C_k with *min_sup*
end for
return $\cup_k L_k$

Pseudo Code of Apriori

Algorithm 2 Apriori

Input: a transaction DB and *min_sup*
Output: all frequent itemsets

$L_1 \leftarrow \{frequent\ items\}$
for $k = 2$; $L_{k-1} \neq \emptyset$; $k \leftarrow k + 1$ **do**
    $C_k \leftarrow$ candidates generated based on L_{k-1} (Candidate Generation)
    Scan Transaction DB to count supports of itemsets in C_k (Counting Supports)
    $L_k \leftarrow$ candidates in C_k with *min_sup*
end for
return $\cup_k L_k$

Candidate Generation in Apriori

- C_k is generated based on L_{k-1}
  - Candidates should be extensions of itemsets in L_{k-1}
- Step 1: Self-joining L_{k-1}
  - Idea: use two $(k-1)$-itemsets in L_{k-1} to make a possibly frequent k-itemset
  - Every itemset is a string in alphabetical order (e.g. items are $a < b < ... < z$, $\{a, d, c, b\} = abcd$)
  - If $I_1[1:k-2] = I_2[1:k-2]$, and $I_1[k-1] < I_2[k-1]$, add $I_3 = I_1 \cup I_2$ to C_k (Prove the completeness by yourself)
- Step 2: Pruning candidates that are supersets of infrequent $(k-1)$-itemsets
  - The anti-monotonicity property of itemsets
  - Check every $(k-1)$-subset of a candidate

An Example of Generating C_k

- $L_3 = \{abc, abd, acd, ace, bcd\}$
- Self-Joining: $L_3 \times L_3$
  - $abcd \leftarrow abc \times abd$
  - $acde \leftarrow acd \times ace$
- Pruning candidates
  - All 3-subsets of $abcd$ are in L_3
  - $acde$ should be pruned since it contains ade which is infrequent
- $C_4 = \{abcd\}$

Bounding #Candidates

- Suppose #frequent_items$= |L_1| = n$ and #frequent_itemsets$= |\cup_{k=1} L_k| = M$
- #Candidates$= |\cup_{k=2} C_k| \leq nM$
- $M = poly(n) \Rightarrow$ #Candidates$=poly(n)$
  - Output sensitive
  - Much better than $O(2^n)$ candidates!
- #frequent_itemsets is sensitive to *min_sup*
  - Challenge: Given *min_sup*, computing #frequent_itemsets is #P-hard

Bounding #Candidates

- Suppose $\#\text{frequent\_items} = |L_1| = n$ and $\#\text{frequent\_itemsets} = |\cup_{k=1} L_k| = M$
- $\#\text{Candidates} = |\cup_{k=2} C_k| \leq nM$
- $M = poly(n) \Rightarrow \#\text{Candidates} = poly(n)$
  - Output sensitive
  - Much better than $O(2^n)$ candidates!
- #frequent_itemsets is sensitive to *min_sup*
  - Challenge: Given *min_sup*, computing #frequent_itemsets is #P-hard

Proof: $I \in L_{k-1}$, $|\{I' \mid I' \in C_k, I' \supseteq I\}| \leq n \Rightarrow |C_k| \leq n|L_{k-1}| \Rightarrow |\cup_{k=2} C_k| = \sum_{k=2} |C_k| \leq \sum_{k=2} n|L_k - 1| = n|\cup_{k=1} L_k| = nM$

A Running Example

min_sup=2

Database TDB

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

A Running Example

Database TDB

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

min_sup=2

C_1

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

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

A Running Example

Database TDB

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

min_sup=2

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

A Running Example

min_sup=2

Database TDB

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

1^{st} scan

C_1

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

L_1

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

C_2

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

A Running Example

min_sup=2

Database TDB

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

1st scan →

C_1

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

L_1

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

C_2

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

2nd scan ←

C_2

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

A Running Example

min_sup=2

Database TDB

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

1[st] scan

C_1

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

L_1

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

C_2

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

2[nd] scan

C_2

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

L_2

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

A Running Example

min_sup=2

Database TDB

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

1[st] scan

C_1

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

L_1

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

C_2

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

2[nd] scan

C_2

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

L_2

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

C_3

Itemset
{B, C, E}

A Running Example

min_sup=2

Database TDB

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

1[st] scan

C_1

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

L_1

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

C_2

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

2[nd] scan

C_2

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

L_2

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

C_3

Itemset
{B, C, E}

3[rd] scan

L_3

Itemset	sup
{B, C, E}	2

Pseudo Code of Apriori

Algorithm 3 Apriori

Input: a transaction DB and *min_sup*
Output: all frequent itemsets

```
L_1 ← {frequent items}
for k = 2; L_{k−1} ≠ ∅; k ← k + 1 do
    C_k ← candidates generated based on L_{k−1} (Candidate Generation)
    Scan Transaction DB to count supports of itemsets in C_k (Counting Supports)
    L_k ← candidates in C_k with min_sup
end for
return ∪_k L_k
```

Counting Supports of Candidate Itemsets

- Scan the transaction DB once to count supports of itemsets in C_k
- Method
  - A hash table (candidates as keys, supports as values)
  - For each transaction, enumerate its k-subsets and increment supports of corresponding itemsets
  - Ignore transactions without any frequent $(k-1)$-itemsets

C_3 ={abc,abd,acd, ace,bcd}

Key	Value
abc	1
abd	3
acd	5
ace	2
bcd	1

3-subsets of trans. acde: acd, ace, ade, cde

Enumerating k-Subsets of a Transaction

- A simple DFS

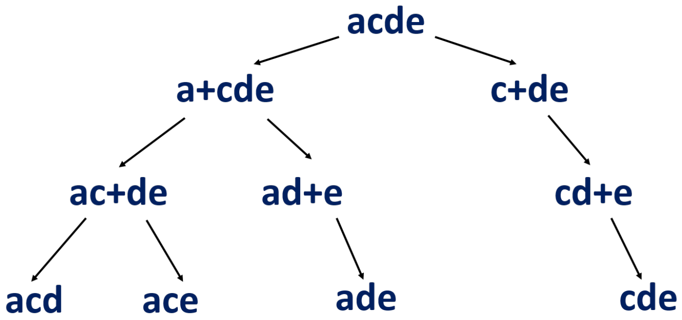
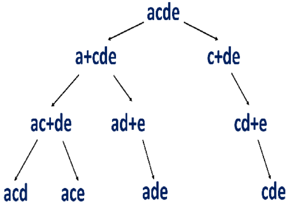
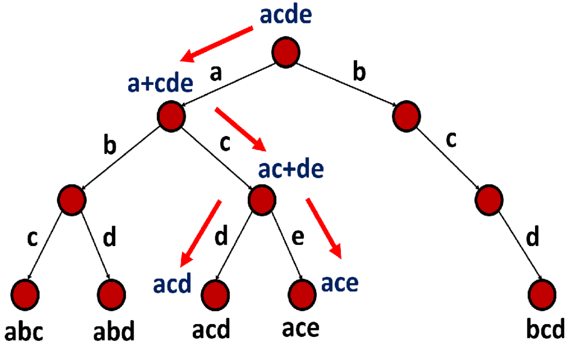


Figure: 3-subsets of the transaction acde

Early Stops Using Prefix Tree (Trie)





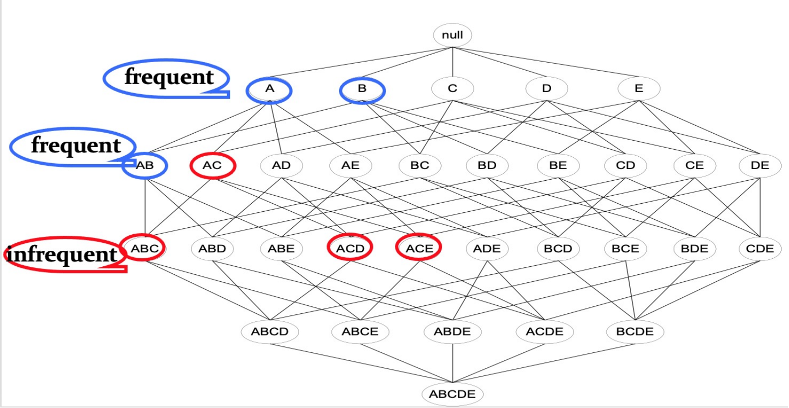
- Store all candidates in a prefix tree
- c+de, ad+e are pruned

Outline

Can We Mine Patterns without Candidate Generation?

- Apriori still may generate too many candidate patterns
  - Reason: Apriori is BFS (breadth-first search)
    - $O(n^k)$ candidates for size-k patterns, where n is the number of frequent items
  - Apriori also needs to scan the DB many times
- Solution
  - Switch to DFS (depth-first search)
  - Compress the DB to reduce the #scan

Apriori as BFS



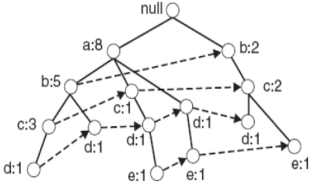
FP-Tree

- Constructing Frequent Pattern Tree (FP-Tree)
  - Scan the DB once and find frequent single items
  - Sort frequent items in frequency descending order, f-list
  - Scan the DB again to store trans in a trie
- The size of an FP-tree is typically smaller than the size of the uncompressed DB because many trans often share a few items in common
  - Best case: all trans have the same set of items, and the FP-tree contains only a single branch of nodes.
  - Worst case: every tran has a unique set of items. As none of the transactions have any items in common, the size of the FP-tree is effectively the same as the size of the DB.

Example of FP-Tree

$a : 8, b : 7, c : 6, d : 5, e : 3$

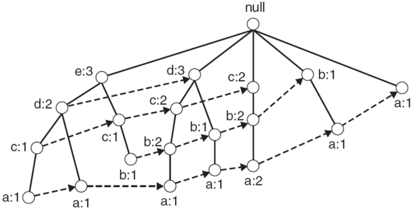
- FP-tree with item descending ordering



Transaction Data Set

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	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}

- FP-tree with item ascending ordering

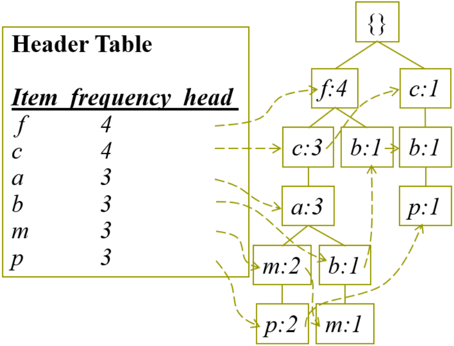


FP-Growth Algorithm

- 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 is more efficient compared to Apriori especially when the DB cannot fit in the memory but the FP-Tree can

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

DFS Decomposition of DB

1	***f, c, a, m, p***
2	***f, c, a, b, m***
3	***f, b***
4	***c, b, p***
5	***f, c, a, m, p***

1	***f, c, a, m***
4	***c, b***
5	***f, c, a, m***

+ *p*

1	***f, c, a, m***
2	***f, c, a, b, m***
3	***f, b***
4	***c, b***
5	***f, c, a, m***

1	***f, c, a***
2	***f, c, a, b***
5	***f, c, a***

+ *m*

1	***f, c, a***
2	***f, c, a, b***
3	***f, b***
4	***c, b***
5	***f, c, a***

2	***f, c, a***
3	***f***
4	***c***

+ *b*

1	***f, c, a***
2	***f, c, a***
3	***f***
4	***c***
5	***f, c, a***

1	***f, c***
2	***f, c***
5	***f, c***

+ *a*

1	***f, c***
2	***f, c***
3	***f***
4	***c***
5	***f, c***

Decomposition of DB

1	***f, c, a, m, p***
2	***f, c, a, b, m***
3	***f, b***
4	***c, b, p***
5	***f, c, a, m, p***

1	***f, c, a, m***
4	***c, b***
5	***f, c, a, m***

+ *p*

(1)

1	***f, c, a***
2	***f, c, a, b***
5	***f, c, a***

+ *m*

(2)

2	***f, c, a***
3	***f***
4	***c***

+ *b*

(3)

1	***f, c***
2	***f, c***
5	***f, c***

+ *a*

(4)

1	***f***
2	***f***
4	
5	***f***

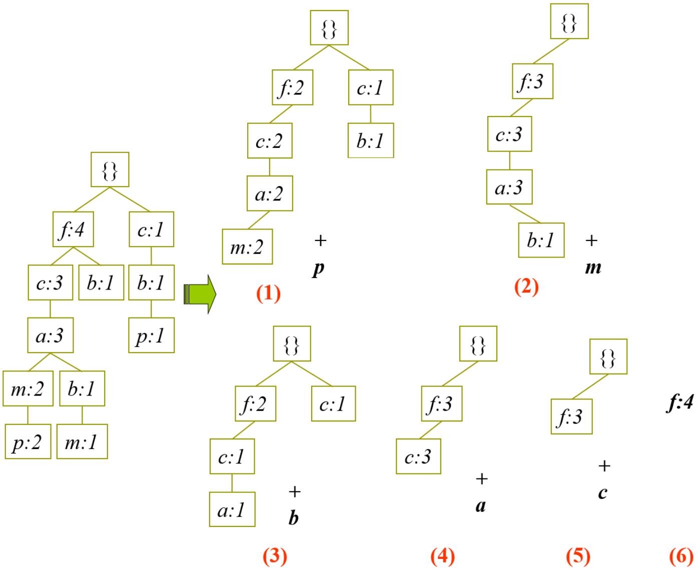
+ *c*

(5)

f: 1,2,3,5

(6)

Decomposition of DB: FP-Tree Perspective



DFS of FP-Growth

min_sup = 3

1	**f, c, a, m, p**
2	**f, c, a, b, m**
3	**f, b**
4	**c, b, p**
5	**f, c, a, m, p**

1	**f, c, a, m**
4	**c, b**
5	**f, c, a, m**

+ p

1	**c**
4	**c**
5	**c**

+ p

p: 3
cp: 3

1	**f, c, a**
2	**f, c, a, b**
5	**f, c, a**

+ m

1	**f, c, a**
2	**f, c, a**
5	**f, c, a**

+ m

m: 3
fm: 3
cm: 3
am: 3
fcm: 3
fam: 3
cam: 3
fcam: 3

2	**f, c, a**
3	**f**
4	**c**

+ b

b: 3

1	**f, c**
2	**f, c**
5	**f, c**

+ a

a: 3
fa: 3
ca: 3
fca: 3

1	**f**
2	**f**
4	
5	**f**

+ c

c: 4
fc: 3

f: 1,2,3,5

f: 4

Outline

Accelerating Apriori: Partition

- A. Savasere *et al.*, An Efficient Algorithm for Mining Association Rules in Large Databases. *VLDB'95*
- Motivation: reducing #scans of transaction DB
  - Scan 1: partition transaction DB and find local frequent itemsets (relative min. support θ)
  - Scan 2: calculate global frequent itemsets
- Correctness: Frequent itemsets must be frequent in at least one partition

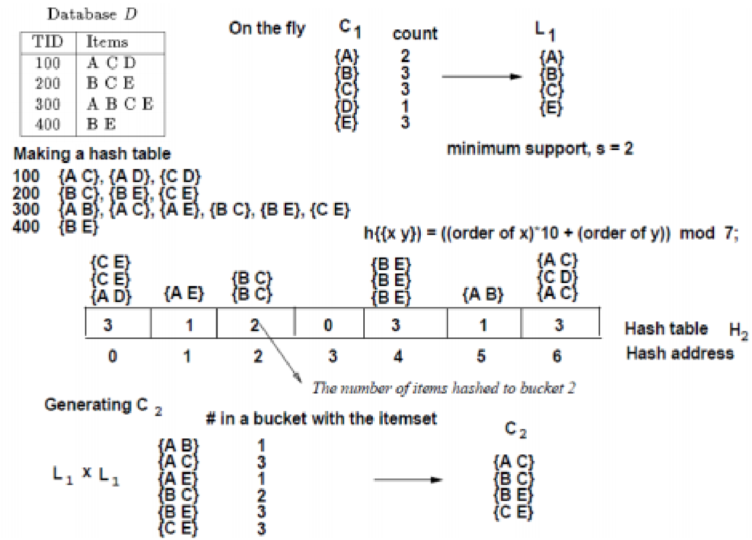
$$\boxed{DB_1} + \boxed{DB_2} + \cdots + \boxed{DB_k} = \text{TBD}$$

$Sup_1(I) < \theta|DB_1|$ $Sup_2(I) < \theta|DB_2|$ $Sup_k(I) < \theta|DB_k|$ $Sup(I) < \theta|TDB|$

Accelerating Apriori: Hashing and Pruning

- J. Park *et al.*, An effective hash-based algorithm for mining association rules. *SIGMOD'95*
- Motivation: reducing #candidates
- Method:
  - At level $k-1$, make a hash table to group different k-itemsets (drew from transactions) in the same bucket
  - #buckets≪#candidates
  - When generating C_k based on L_{k-1}, if I is in an infrequent bucket, remove it from C_k
- Especially effective for reducing C_2

Hashing and Pruning: An Example



Accelerating FPM: Sampling

- M. Riondato *et al.*, Efficient Discovery of Association Rules and Frequent Itemsets through Sampling with Tight Performance Guarantees, *ECML PKDD'12*
- Only sample $O(\frac{1}{\epsilon^2}(D + \log \frac{1}{\delta}))$ transactions
  - $E[Freq(I; S)] = Freq(I)$
  - $L = \{I \mid Freq(I; S) \geq T - \frac{\epsilon}{2}\}$ ($T = \frac{min\_sup}{|TBD|}$)
  - D is decided by the transaction DB, usually very small
    - If TDB does not have many long transactions
    - Sample size is a constant to #transactions
  - All sampled transactions can fit into main memory!
- Tolerable errors of L (with high probability $1 - \delta$)
  - All real frequent patterns can be found
  - If I is not frequent, $Freq(I) \geq T - \epsilon$

Outline

Advanced Frequent Pattern Mining

- FPM in stream Transaction DB
  - Transactions are arriving continuously
  - Incrementally maintain FP in the full/recent TDB
- Complex types of patterns: sequential patterns, subgraph patterns, ...
  - The idea of Apriori still works
- Reference: "**Frequent Pattern Mining**", Aggarwal, Charu C., Han, Jiawei

Outline

Frequent Patterns: Basic Concepts

Frequent Itemset/Pattern Mining in Real Applications

Frequent Itemset/Pattern Mining Algorithms
- Challenges
- Apriori Algorithm
- Pattern Growth Algorithm
- Accelerations
- Advanced FPM

Conclusion

Conclusion

- Frequent Pattern Mining
  - Basic concepts and why it is important
- Examples of Frequent Itemsets Mining
  - Association rules market-basket analysis
  - Sub-market finding in Sponsored Search
- FPM Algorithms
  - Apriori: effective pruning based on anti-monotonicity
  - Pattern Growth: DFS without candidate generation
  - Accelerations
- Readings
  - Chapter 6 of "Data Mining: Concepts and Techniques"
  - Chapter 6 of "Mining of Massive Datasets"