FP-growth

- Challenges of Frequent Pattern Mining
- Improving Apriori
- Fp-growth
  - Fp-tree
  - Mining frequent patterns with FP-tree
- Visualization of Association Rules

Challenges of Frequent Pattern Mining

- Challenges
  - Multiple scans of transaction database
  - Huge number of **candidates**
  - Tedious workload of support counting for candidates
- Improving Apriori: general ideas
  - Reduce passes of transaction database scans
  - **Shrink number of candidates**
  - Facilitate support counting of candidates

Transactional Database

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

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

Association Rule Mining

- Find all frequent itemsets
- Generate strong association rules from the frequent itemsets

- Apriori algorithm is mining frequent itemsets for Boolean associations rules

Improving Apriori

- Reduce passes of transaction database scans
- **Shrink number of candidates**
- Facilitate support counting of candidates
- Use constraints

The Apriori Algorithm — Example

Database D

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

Scan D

C_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{4}	1
{5}	3

L_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{5}	3

C_2

itemset
{1 2}
{1 3}
{1 5}
{2 3}
{2 5}
{3 5}

Scan D

C_2

itemset	sup
{1 2}	1
{1 3}	2
{1 5}	1
{2 3}	2
{2 5}	3
{3 5}	2

L_2

itemset	sup
{1 3}	2
{2 3}	2
{2 5}	3
{3 5}	2

C_3

itemset
{2 3 5}

Scan D

L_3

itemset	sup
{2 3 5}	2

Apriori + Constraint

Database D

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

Scan D

C_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{4}	1
{5}	3

L_1

itemset	sup.
{1}	2
{2}	3
{3}	3
~~{5}~~	~~3~~

C_2

itemset
{1 2}
{1 3}
{1 5}
{2 3}
{2 5}
{3 5}

Scan D

C_2

itemset	sup
{1 2}	1
{1 3}	2
{1 5}	1
{2 3}	2
{2 5}	3
{3 5}	2

L_2

itemset	sup
{1 3}	2
~~{2 3}~~	~~2~~
~~{2 5}~~	~~3~~
~~{3 5}~~	~~2~~

C_3

itemset
{2 3 5}

Scan D

L_3

itemset	sup
~~{2 3 5}~~	~~2~~

Constraint:

Sum{S.price} < 5

Push an Anti-monotone Constraint Deep

Database D

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

Scan D

C_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{4}	1
~~{5}~~	~~3~~

L_1

itemset	sup.
{1}	2
{2}	3
{3}	3
~~{5}~~	~~3~~

C_2

itemset
{1 2}
{1 3}
~~{1 5}~~
{2 3}
~~{2 5}~~
~~{3 5}~~

Scan D

C_2

itemset	sup
{1 2}	1
{1 3}	2
~~{1 5}~~	~~1~~
~~{2 3}~~	~~2~~
~~{2 5}~~	~~3~~
~~{3 5}~~	~~2~~

L_2

itemset	sup
{1 3}	2
~~{2 3}~~	~~2~~
~~{2 5}~~	~~3~~
~~{3 5}~~	~~2~~

C_3

itemset
~~{2 3 5}~~

Scan D

L_3

itemset	sup
~~{2 3 5}~~	~~2~~

Constraint:

Sum{S.price} < 5

Hash-based technique

- The basic idea in hash coding is to determine the address of the stored item as some simple arithmetic function content
- Map onto a subspace of allocated addresses using a hash function
- Assume the allocated address range from b to $n+b-1$, the hashing function may take $h=(a \bmod n)+b$
- In order to create a good pseudorandom number, n ought to be prime

---

- Two different keywords may have equal hash addresses
- Partition the memory into buckets, and to address each bucket
  - One address is mapped into one bucket

- When scanning each transition in the database to generate frequent 1-itemsets, we can generate all the 2-itemsets for each transition and hash them into different buckets of the hash table
- We use *h=a mod n, a* address, *n* < the size of C_2

- A 2-itemset whose bucket count in the hash table is below the support threshold cannot be frequent, and should be removed from the candidate set

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

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

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}

Transaction reduction

- A transaction which does not contain frequent k-itemsets should be removed from the database for further scans

Partitioning

- First scan:
  - Subdivide the transactions of database D into **n** non overlapping partitions
  - If the minimum support in D is *min_sup*, then the minimum support for a partition is *min_sup * number of transactions in that partition*
  - Local frequent items are determined
  - A local frequent item my not by a frequent item in D
- Second scan:
  - Frequent items are determined from the local frequent items

Partitioning

- First scan:
  - Subdivide the transactions of database D into n non overlapping partitions
  - If the minimum support in D is *min_sup*, then the minimum support for a partition is

    *min_sup * number of transactions in D / number of transactions in that partition*

  - Local frequent items are determined
  - A local frequent item my not by a frequent item in D
- Second scan:
  - Frequent items are determined from the local frequent items

Sampling

- Pick a random sample S of D
- Search for local frequent items in S
  - Use a lower support threshold
  - Determine frequent items from the local frequent items
  - Frequent items of D may be missed
- For completeness a second scan is done

Is Apriori fast enough?

- Basics of Apriori algorithm
  - Use frequent (k-1)-itemsets to generate k-itemsets candidates
  - Scan the databases to determine frequent k-itemsets

- It is costly to handle a huge number of candidate sets
- If there are 10^4 frequent *1-itemsts*, the Apriori algorithm will need to generate more than 10^7 *2-itemsets* and test their frequencies

- To discover a 100-itemset
- 2^{100}-1 candidates have to be generated

$$2^{100}-1=1.27*10^{30}$$

(Do you know how big this number is?)
....

- $7*10^{27}$ ≈ number of atoms of a person
- $6*10^{49}$ ≈ number of atoms of the earth
- 10^{78} ≈ number of the atom of the universe

Bottleneck of Apriori

- Mining long patterns needs many passes of scanning and generates lots of candidates
- Bottleneck: **candidate-generation-and-test**
- Can we avoid **candidate generation**?
- May some new data structure help?

Mining Frequent Patterns Without *Candidate* Generation

- Grow long patterns from short ones using local frequent items
  - "abc" is a frequent pattern
  - Get all transactions having "abc": DB|abc
  - "d" is a local frequent item in DB|abc → abcd is a frequent pattern

Construct FP-tree from a Transaction Database

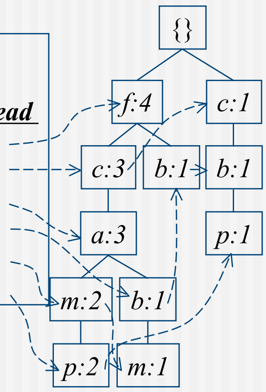
TID	*Items bought*	*(ordered) frequent items*
100	*{f, a, c, d, g, i, m, p}*	*{f, c, a, m, p}*
200	*{a, b, c, f, l, m, o}*	*{f, c, a, b, m}*
300	*{b, f, h, j, o, w}*	*{f, b}*
400	*{b, c, k, s, p}*	*{c, b, p}*
500	*{a, f, c, e, l, p, m, n}*	*{f, c, a, m, p}*

min_support = 3

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

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

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)
  - There exists examples of databases, where compression ratio could be over 100

- The size of the FP-trees bounded by the overall occurrences of the frequent items in the database
- The height of the tree is bound by the maximal number of frequent items in a transaction

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

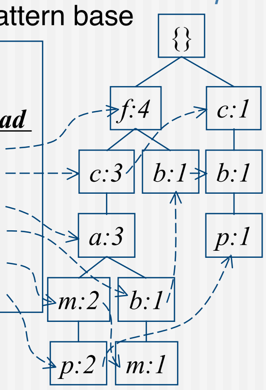
  Pattern f
- Completeness and non-redundency

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

Header Table

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



***Conditional* pattern bases**

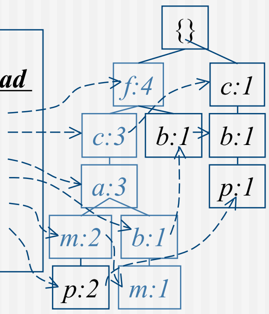
item	*cond. pattern base*
c	***f:3***
a	***fc:3***
b	***fca:1, f:1, c:1***
m	***fca:2, fcab:1***
p	***fcam:2, cb:1***

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

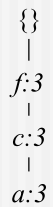
Header Table

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



m-conditional **pattern base:**
fca:2, fcab:1

➔

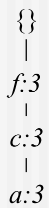


m-conditional **FP-tree**

➔

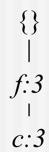
All frequent patterns relate to *m*
m,
fm, cm, am,
fcm, fam, cam,
fcam
-> associations

Recursion: Mining Each Conditional FP-tree



m-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**

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

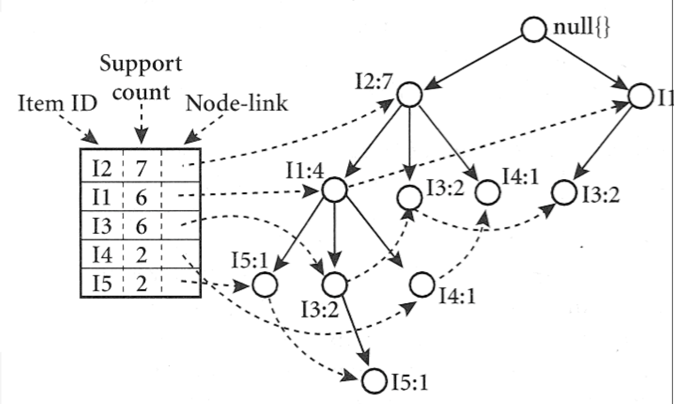
cam-conditional **FP-tree**

item	conditional pattern base	conditional FP-tree
p	{(fcam:2), (cb:1)}	{(c:3)}|p
m	{(fca:2), (fcab:1)}	{(f:3, c:3, a:3)}|m
b	{(fca:1), (f:1), (c:1)}	leer
a	{(fc:3)}	{(f:3, c:3)}|a
c	{(f:3)}	{(f:3)}|c
f	leer	leer

Mining Frequent Patterns With FP-trees

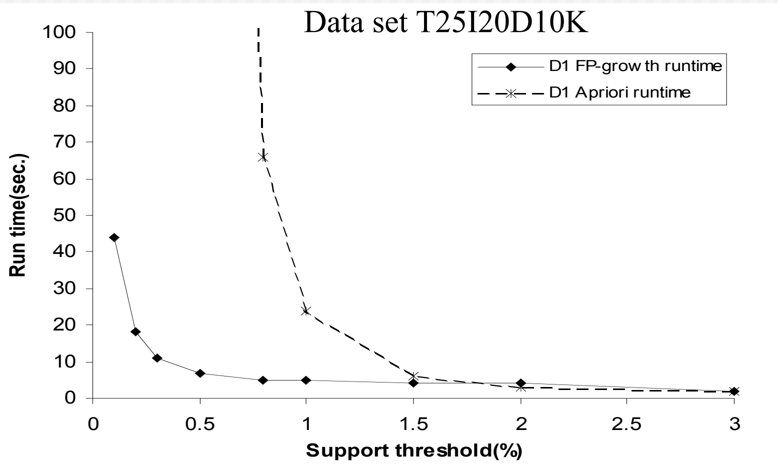
- Idea: Frequent pattern growth
  - Recursively grow frequent patterns by pattern and database partition
- Method
  - For each frequent item, construct its conditional pattern-base, and then its conditional FP-tree
  - Repeat the process on each newly created conditional FP-tree
  - Until the resulting 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

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



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: 2, I2 I1 I3: 2
I1	{(I2: 4)}	⟨I2: 4⟩	I2 I1: 4

Experiments: FP-Growth vs. Apriori

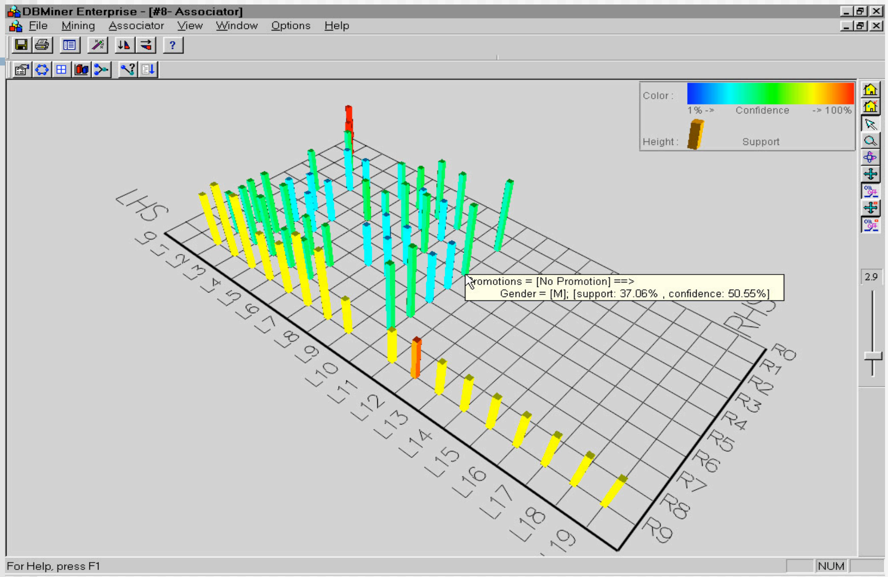


- Advantage when support decrease
- No prove
  - advantage is shown by experiments with artificial data

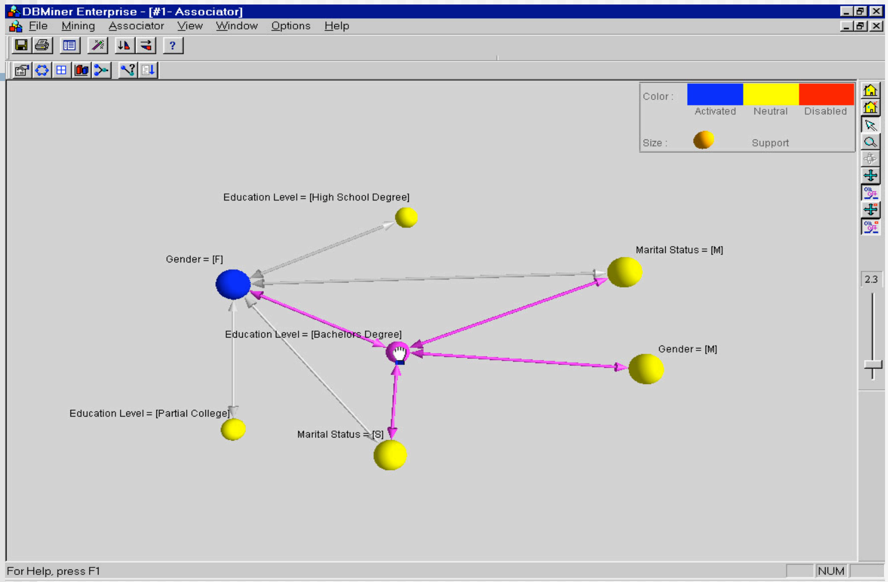
Advantages of FP-Growth

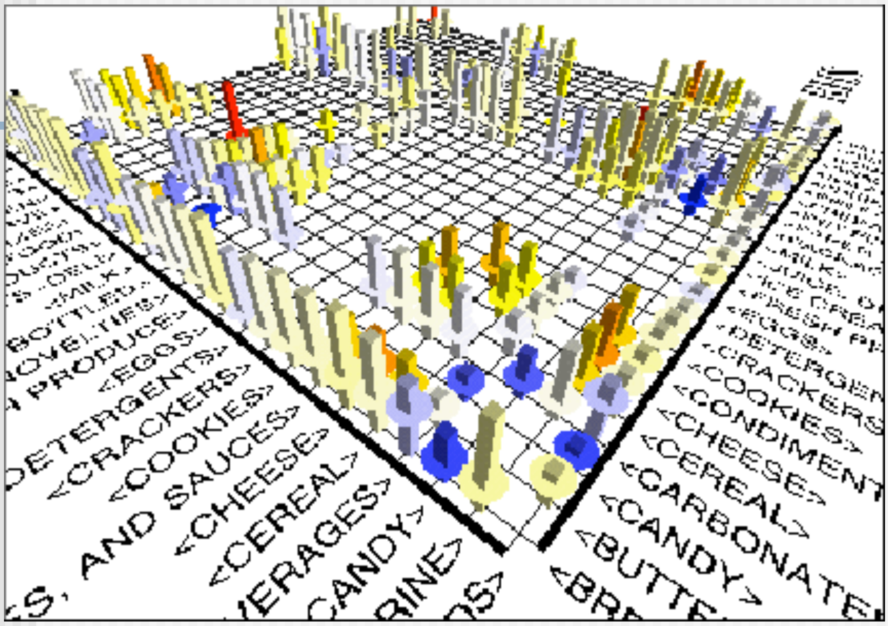
- Divide-and-conquer:
  - decompose both the mining task and DB according to the frequent patterns obtained so far
  - leads to focused search of smaller databases
- Other factors
  - no candidate generation, no candidate test
  - compressed database: FP-tree structure
  - no repeated scan of entire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

Visualization of Association Rules: Plane Graph



Visualization of Association Rules: Rule Graph





- Challenges of Frequent Pattern Mining
- Improving Apriori
- Fp-growth
  - Fp-tree
  - Mining frequent patterns with FP-tree
- Visualization of Association Rules

- Clustering
- k-means, EM