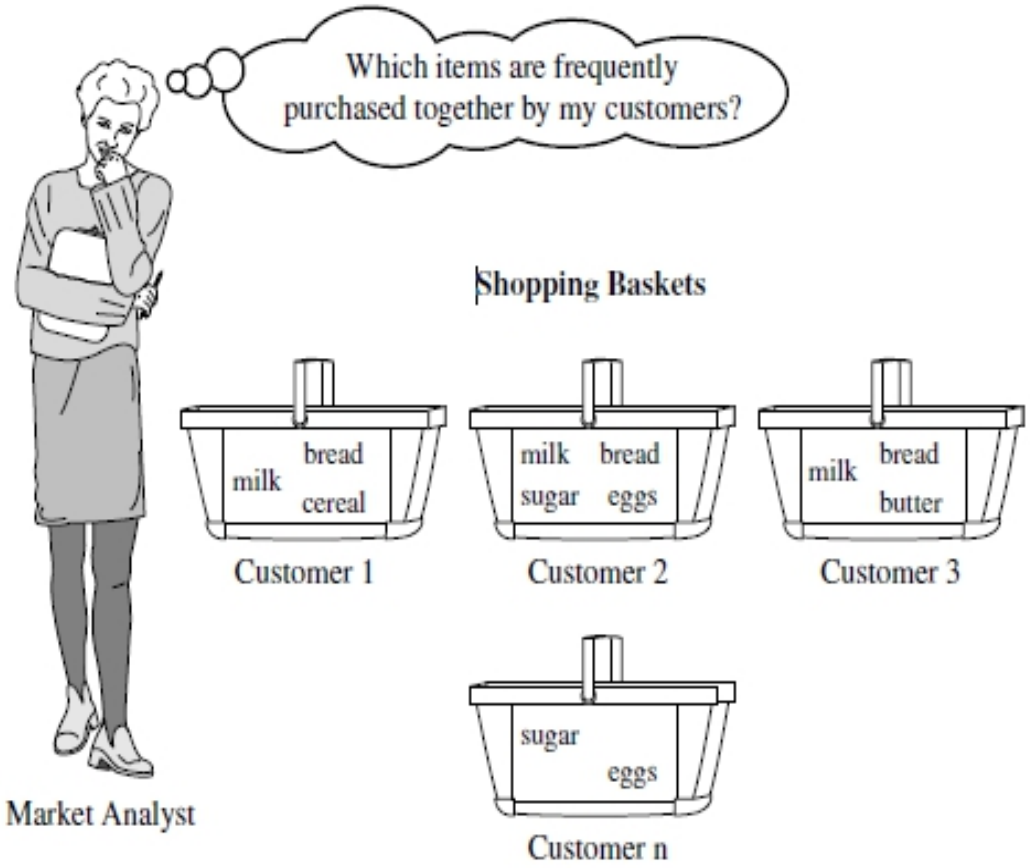
Contents(Unit-IV)

- Mining frequent patterns and Association Rules
- Market basket analysis
- Frequent item sets and association rules
- Apriori algorithm
- FP growth algorithm
- Improving efficiency of Apriori and FP growth algorithms.
- https://docs.google.com/forms/d/e/1FAIpQLSftQJPWlTBRyq3iXuS-tApOE1nCVJx8NN4YJtqnYhy8Njhr7Q/viewform?usp=pp_url

Introduction

- **Frequent patterns** are patterns (such as itemsets, subsequences, or substructures) that appear in a data set frequently.
- For example, a set of items, such as milk and bread, that appear frequently together in a transaction data set is a **frequent itemset**.
- A subsequence, such as buying first a PC, then a digital camera, and then a memory card, if it occurs frequently in a shopping history database, is a **(frequent) sequential pattern**.
- A substructure can refer to different structural forms, such as subgraphs, subtrees, or sublattices, which may be combined with itemsets or subsequences.
- If a substructure occurs frequently, it is called a **(frequent) structured pattern.**
- Finding such frequent patterns plays an essential role in mining associations, correlations, and many other interesting relationships among data.

Market Basket Analysis



What is Frequent Itemset Mining?

Frequent Itemset Mining:

Finding frequent patterns, associations, correlations, or causal structures among sets of items or objects in transaction databases, relational databases, and other information repositories.

- Given:
  - –A set of items $I=\{i1,i2,\ldots,im\}$
  - –A database of transactions D, where a transaction $T \subseteq I$ is a set of items
- Task 1:find all subsets of items that occur together in many transactions.
  - –E.g.: 85% of transactions contain the itemset{milk, bread, butter}
- Task 2:find all rules that correlate the presence of one set of items with that of another set of items in the transaction database.
  - –E.g.: 98% of people buying tires and auto accessories also get automotive service done
- Applications: Basket data analysis, cross-marketing, catalog design, loss-leader analysis, clustering, classification, recommendation systems, etc.

Transactional 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

Definitions:
An *item*: an article in a basket, or an attribute-value pair
A *transaction*: set of items purchased in a basket; it may have TID (transa
A *transactional dataset*: A set of transactions

Itemsets & Frequent Itemsets

- itemset: A set of one or more items
- k-itemset X = {x1, …, xk}
- *(absolute) support*, or, *support count* of X:
- Frequency or occurrence of an itemset X
- *(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

Association Rule

What is an association rule?

- An implication expression of the form X Y, where X and Y are itemsets and X Y=

- Example:
  {Milk, Diaper} {Beer}

Association Rule Mining

- To find all the strong association rules
- An association rule r is strong if
  - Support(r) ≥ min_sup
  - Confidence(r) ≥ min_conf
- Rule Evaluation Metrics
  - Support (s): Fraction of transactions that contain both X and Y

$$P(X \cup Y) = \frac{\#\, trans\ containing\ (X \cup Y)}{\#\, trans\ in\ D}$$

= support count (X U Y) / number of all transactions

  - Confidence (c): Measures how often items in Y appear in transactions that contain X

$$P(X \mid Y) = \frac{\#\, trans\ \ containing\ \ \ (X \cup Y)}{\#\, trans\ \ containing\ \ \ X}$$

= support count (X U Y) / support count (X)

Example of Support and Confidence

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

To calculate the support and confidence of rule {Milk, Diaper} {Beer}

- # of transactions: 5
- # of transactions containing {Milk, Diaper, Beer}: 2
- Support: 2/5=0.4

- # of transactions containing {Milk, Diaper}: 3
- Confidence: 2/3=0.67

Definition: Frequent Itemset

- Itemset
  - A collection of one or more items
    - Example: {Bread, Milk, Diaper}
  - k-itemset
    - An itemset that contains k items
- Support count ( )
  - # transactions containing an itemset
  - E.g. ({Bread, Milk, Diaper}) = 2
- Support (s)
  - Fraction of transactions containing an itemset
  - E.g. s({Bread, Milk, Diaper}) = 2/5
- Frequent Itemset
  - An itemset whose support is greater than or equal to a *min_sup* threshold

Association Rule Mining Task

- An association rule r is strong if
  - Support(r) ≥ min_sup
  - Confidence(r) ≥ min_conf

- Given a transactions database D, the goal of association rule mining is to find all strong rules

- Two-step approach:
  1. Frequent Itemset Identification
     - Find all itemsets whose support min_sup
  2. Rule Generation
     - From each frequent itemset, generate all confident rules whose confidence min_conf

Rule Generation

Suppose min_sup=0.3, min_conf=0.6,
Support({Beer, Diaper, Milk})=0.4

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

All candidate rules:

{Beer} {Diaper, Milk} (s=0.4, c=0.67)
{Diaper} {Beer, Milk} (s=0.4, c=0.5)
{Milk} {Beer, Diaper} (s=0.4, c=0.5)
{Beer, Diaper} {Milk} (s=0.4, c=0.67)
{Beer, Milk} {Diaper} (s=0.4, c=0.67)
{Diaper, Milk} {Beer} (s=0.4, c=0.67)

All non-empty real subsets

{Beer} , {Diaper} , {Milk}, {Beer, Diaper}, {Beer, Milk} , {Diaper, Milk}

Strong rules:

{Beer} {Diaper, Milk} (s=0.4, c=0.67)
{Beer, Diaper} {Milk} (s=0.4, c=0.67)
{Beer, Milk} {Diaper} (s=0.4, c=0.67)
{Diaper, Milk} {Beer} (s=0.4, c=0.67)

Association Rule Mining

- Major steps in association rule mining
  - Frequent itemsets computation
  - Rule derivation

- Use of support and confidence to measure strength

Scalable Methods for Mining Frequent Patterns

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-and-Test Approach

- A *frequent* (used to be called large) *itemset* is an itemset whose support (S) is ≥ minSup.

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

- **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

Implementation of Apriori

- How to generate candidates?
  - Step 1: self-joining L_k
  - Step 2: pruning
- How to count supports of candidates?
- 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}

How to Generate Candidates?

- Suppose the items in L_{k-1} are listed in an order
- Step 1: self-joining L_{k-1}

  insert into $\mathbf{C_k}$

  select $\mathbf{p.item_1, p.item_2, \ldots, p.item_{k-1}, q.item_{k-1}}$

  from $\mathbf{L_{k-1}\ p, L_{k-1}\ q}$

  where $\mathbf{p.item_1=q.item_1, \ldots, p.item_{k-2}=q.item_{k-2}, p.item_{k-1} < q.item_{k-1}}$
- Step 2: pruning

  forall **itemsets c in** $\mathbf{C_k}$ do

  forall **(k-1)-subsets s of c** do

  if (s is not in L_{k-1}) **then delete** c **from** C_k

Self-joining

- Lk is generated by joining Lk-1 with itself.
- Apriori assumes that items within a transaction or itemset are sorted in lexicographic order.
- l1 from Lk-1 = {l1[1], l1[2],…., l1[k-2], l1[k-1] }
- l2 from Lk-1 = {l2[1], l2[2],…., l2[k-2], l2[k-1] }
- Only when first **k-2** items of l1 and l2 are in common, and l1[k-1] < l2[k-1], these two itemsets are joinable

Generating Candidates

Example 1:

- L3= {(ACF), (ACG), (AFG), (AFH), (CFG)}
- Candidates after the join step: {(ACFG), (AFGH)}
- In the pruning step: delete(AFGH) because
  (FGH) ∉ L3, i.e., (FGH) is not a frequent 3-itemset;
  also (AGH) ∉ L3
- C4= {(ACFG)} → check the support to generate L4

The Apriori Algorithm

- Pseudo-code:

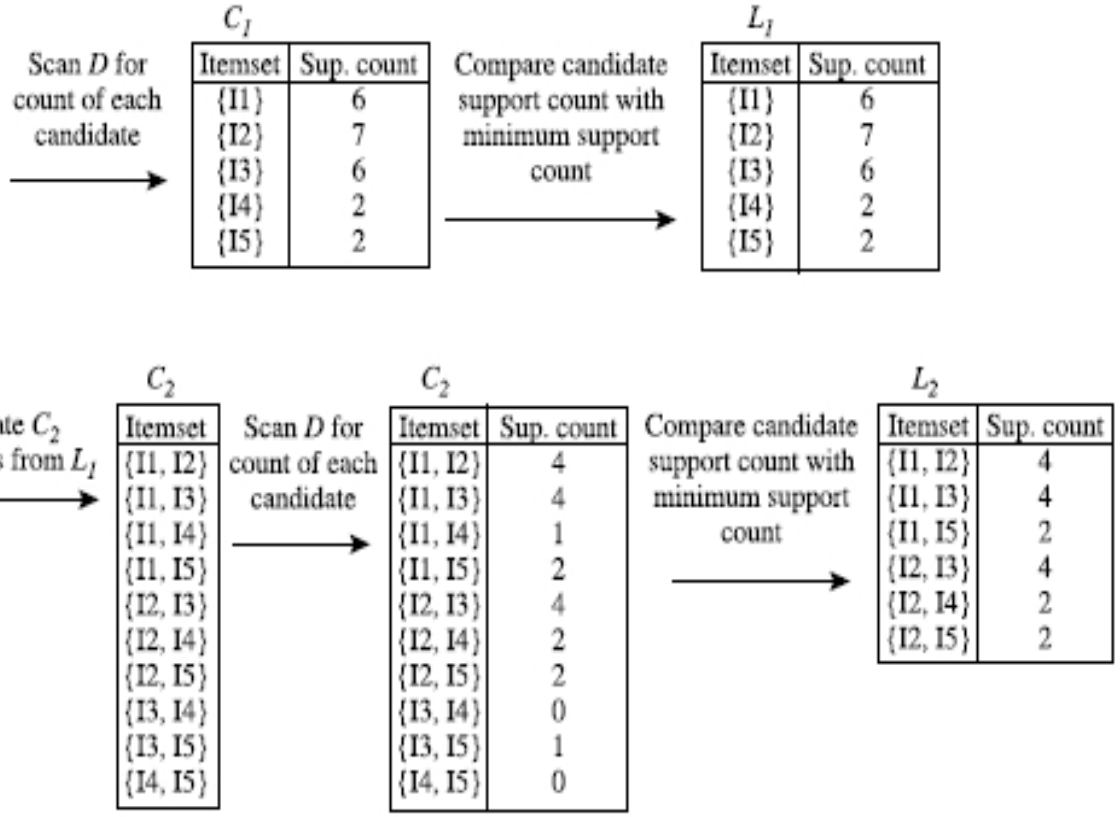
C_k: Candidate itemset of size k
L_k : frequent itemset of size k

L_1 = {frequent items};
for (k = 1; L_k != ; k++) **do begin**
C_{k+1} = candidates generated from L_k;
for each transaction t in database do
increment the count of all candidates in C_{k+1} that are contained in t
L_{k+1} = candidates in C_{k+1} with min_support
end
return $_k L_k$;

Example1

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

Let’s look at a concrete example, based on the *AllElectronics* transaction database, D, There are nine transactions in this database, that is, $|D| = 9$. The Apriori algorithm is used for finding frequent itemsets in D.


C_1
Scan D for count of each candidate
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
C_2
Generate C_2 candidates from L_1
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
C_3
Generate C_3 candidates from L_2
Itemset
{I1, I2, I3}
Scan D for count of each candidate
C_3
Itemset | Sup. count
{I1, I2, I3} 2
Compare candidate support count with minimum support count
L_3
Itemset | Sup. count
{I1, I2, I3} 2

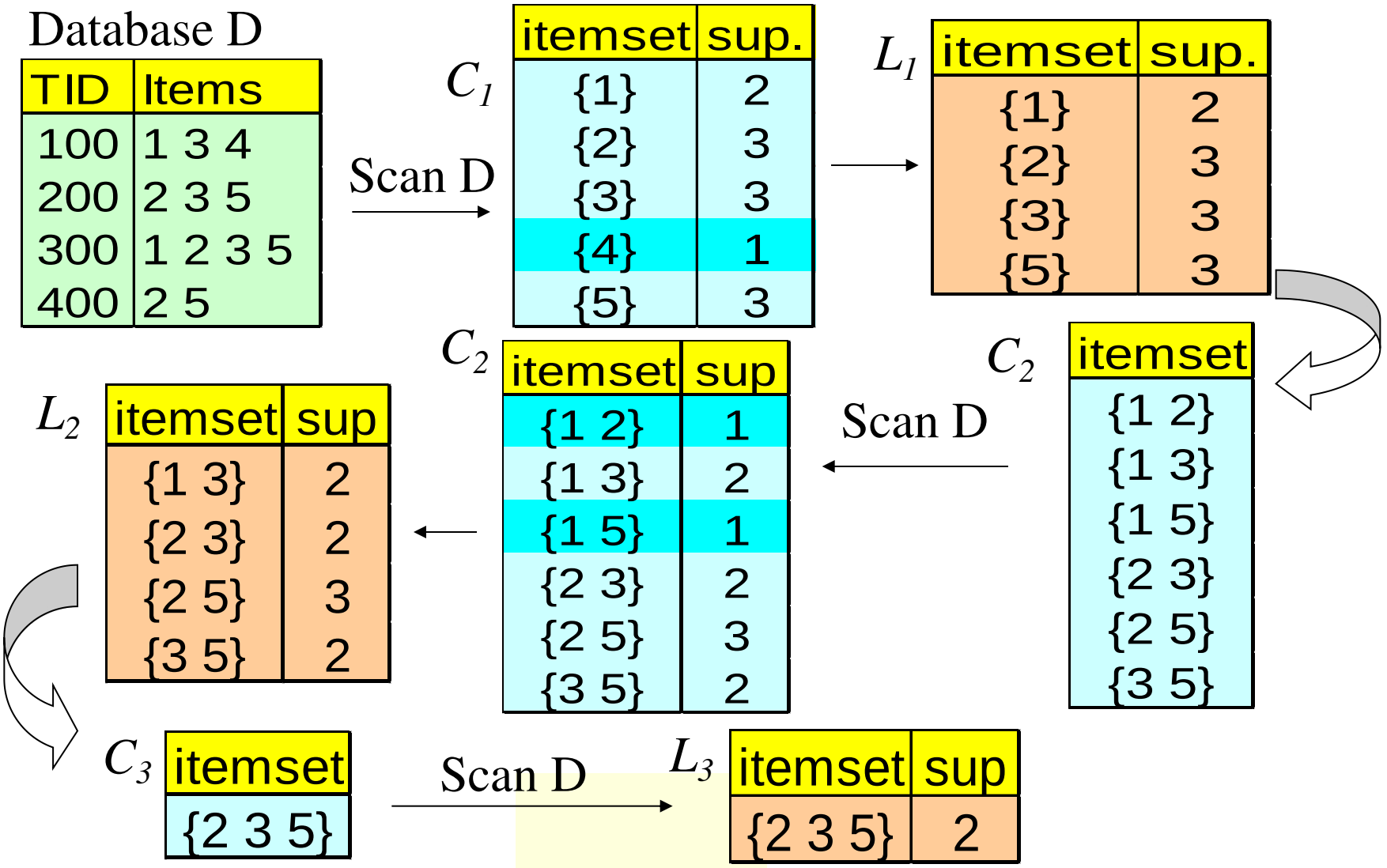
- After self join

{I1,I2,I3,I5}

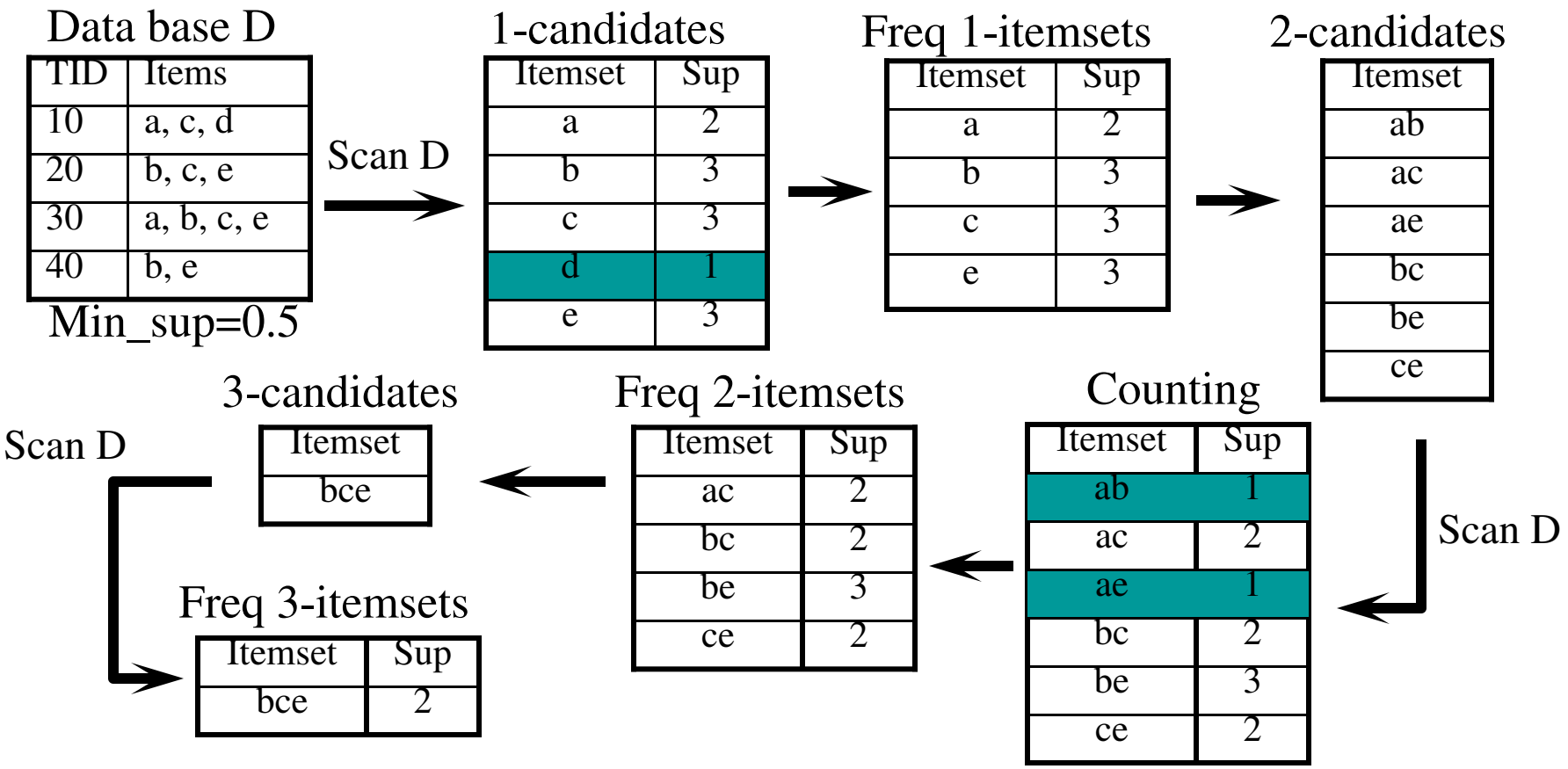
Pruning

C4={}

The Apriori Algorithm — Example



Apriori-based Mining



Challenges of Apriori Algorithm

- Challenges
  - Multiple scans of transaction database
  - Huge number of candidates
  - Tedious workload of support counting for candidates

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

Further Improvement of the Apriori Method

- Major computational 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

- Completeness: any association rule mining algorithm should get the same set of frequent itemsets.

Generating Association Rules from Frequent Itemsets

- Frequent itemsets != association rules
- One more step is required to find 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,

    where support (A B) = support (AUB) and

    confidence (A B) = support _count(AUB) / support_count(A)
- .

Example

- Suppose the data contain the frequent itemset $L = \{I1, I2, I5\}$.
- The association rules that can be generated from L …
- nonempty subsets of L are {I1, I2},{I1, I5}, {I2, I5}, {I1}, {I2}, and {I5}.
- The resulting association rules are as shown below, each listed with its confidence:
  - *I*1^*I*2 *I*5, *confidence* = 2/4 = 50%
  - *I*1^*I*5 *I*2, *confidence* = 2/2 = 100%
  - *I*2^*I*5) *I*1, *confidence* = 2/2 = 100%
  - *I*1 *I*2^*I*5, *confidence* = 2/6 = 33%
  - *I*2 *I*1^*I*5, *confidence* = 2/7 = 29%
  - *I*5 *I*1^*I*2, *confidence* = 2/2 = 100%
- If the minimum confidence threshold is, say, 70%, then only the second, third, and last rules above are output, because these are the only ones generated that are strong.
- Note that, unlike conventional classification rules, association rules can contain more than one conjunct in the right-hand side of the rule..

Improving Apriori's Efficiency

- Hash-based itemset counting: A *k*-itemset whose corresponding hashing bucket count is below the threshold cannot be frequent
- Transaction reduction: A transaction that does not contain any frequent k-itemset is useless in subsequent scans
- Partitioning: Any itemset that is potentially frequent in DB must be frequent in at least one of the partitions of DB
- Sampling: mining on a subset of given data, need a lower support threshold + a method to determine the completeness
- Dynamic itemset counting: add new candidate itemsets immediately (unlike Apriori) when all of their subsets are estimated to be frequent

Is Apriori Fast Enough? — Performance Bottlenecks

- The core of the Apriori algorithm:
  - Use frequent (k – 1)-itemsets to generate candidate frequent k-itemsets
  - Use database scan and pattern matching to collect counts for the candidate itemsets
- The bottleneck of *Apriori*: candidate generation
  - Huge candidate sets:
    - 10^4 frequent 1-itemset will generate 10^7 candidate 2-itemsets
    - To discover a frequent pattern of size 100, e.g., $\{a_1, a_2, \ldots, a_{100}\}$, one needs to generate 2^{100} 10^{30} candidates.
  - Multiple scans of database:
    - Needs (n +1) scans, n is the length of the longest pattern

Mining Frequent Patterns Without Candidate generation

- 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
- Develop an efficient, FP tree based frequent pattern mining method
  - A divide and conquer methodology: decompose mining tasks into small ones
  - Avoid candidate generation: sub database test only
- Idea:
  - Compress database into FP tree, retaining the itemset association information
  - Divide the compressed database into conditional databases, each associated with one frequent item and mine each such database separately.

Frequent Pattern Growth Algorithm

- To overcome these redundant steps, a new association-rule mining algorithm was developed named Frequent Pattern Growth Algorithm. It overcomes the disadvantages of the Apriori algorithm by storing all the transactions in a Trie Data Structure.

- Consider the following data:-

Transaction ID	Items
T1	{E,K,M,N,O,Y}
T2	{D,E,K,N,O,Y}
T3	{A,E,K,M}
T4	{C,K,M,U,Y}
T5	{C,E,I,K,O,O}

The frequency of each individual item is computed.

Item	Frequency
A	1
C	2
D	1
E	4
I	1
K	5
M	3
N	2
O	3
U	1
Y	3

Frequent Pattern set

- Let the minimum support be 3.
- A **Frequent Pattern set** is built which will contain all the elements whose frequency is greater than or equal to the minimum support.
- These elements are stored in descending order of their respective frequencies.
- After insertion of the relevant items, the set L looks like this:-

L = {K : 5, E : 4, M : 3, O : 3, Y : 3}

Ordered-Item set

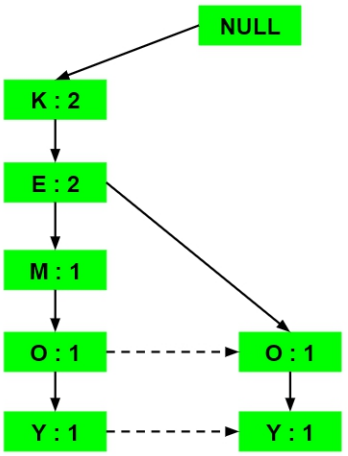
- For each transaction, the respective **Ordered-Item set** is built.
- It is done by iterating the Frequent Pattern set and checking if the current item is contained in the transaction in question.
- **L = {K : 5, E : 4, M : 3, O : 3, Y : 3}**
- If the current item is contained, the item is inserted in the Ordered-Item set for the current transaction.
- The following table is built for all the transactions:-

Transaction ID	Items	Ordered-Item Set
T1	{E,K,M,N,O,Y}	{K,E,M,O,Y}
T2	{D,E,K,N,O,Y}	{K,E,O,Y}
T3	{A,E,K,M}	{K,E,M}
T4	{C,K,M,U,Y}	{K,M,Y}
T5	{C,E,I,K,O,O}	{K,E,O}

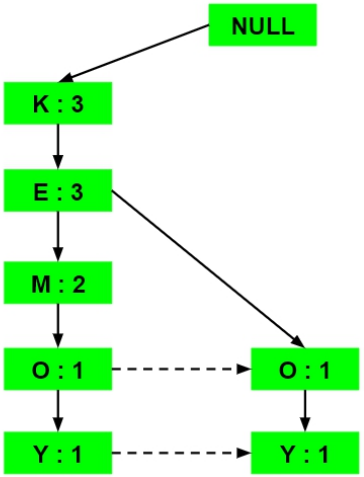
FP-Tree

A) **Inserting the set {K, E, M, O, Y}:**

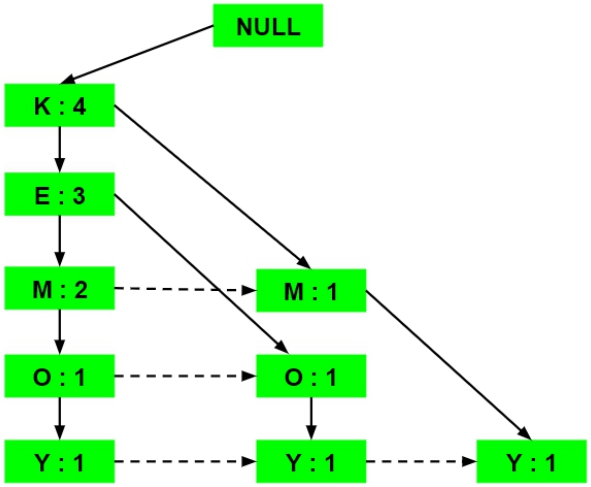
Inserting the set {K, E, O, Y}:



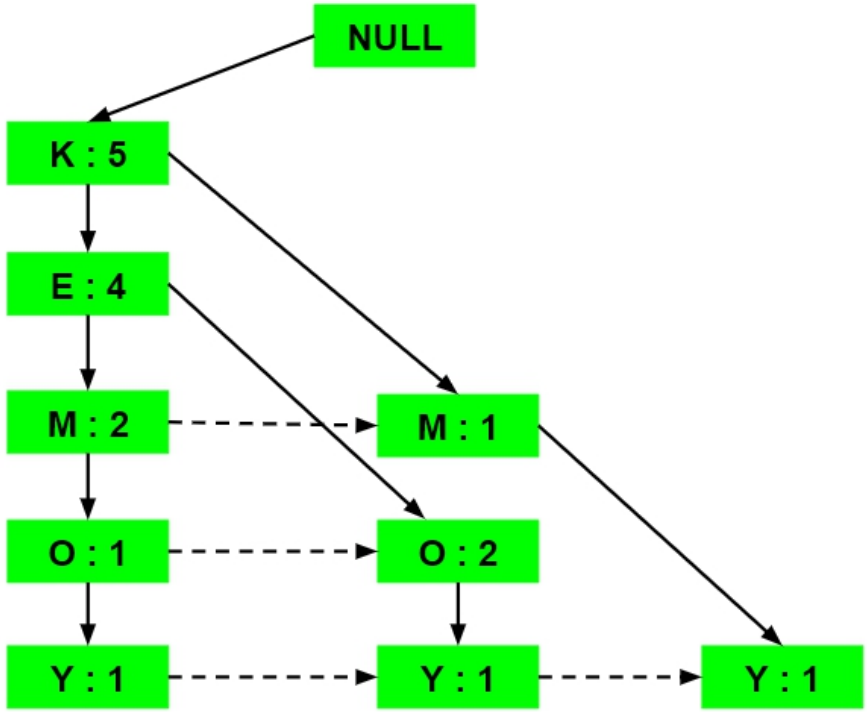
c) **Inserting the set {K, E, M}:**



d)**Inserting the set {K, M, Y}:**



d)**Inserting the set {K, E, O}:**



Conditional Pattern Base

- For each item, the **Conditional Pattern Base** is computed which is path labels of all the paths which lead to any node of the given item in the frequent-pattern tree.
- Note that the items in the below table are arranged in the ascending order of their frequencies.

Items	Conditional Pattern Base
Y	{{K,E,M,O : 1}, {K,E,O : 1}, {K,M : 1}}
O	{{K,E,M : 1}, {K,E : 2}}
M	{{K,E : 2}, {K : 1}}
E	{K : 4}
K	

Conditional Frequent Pattern Tree

- For each item the **Conditional Frequent Pattern Tree is built.**
- It is done by taking the set of elements which is common in all the paths in the Conditional Pattern Base of that item and calculating it's support count by summing the support counts of all the paths in the Conditional Pattern Base.

Items	Conditional Pattern Base	Conditional Frequent Pattern Tree
Y	{{K,E,M,O : 1}, {K,E,O : 1}, {K,M : 1}}	{K : 3}
O	{{K,E,M : 1}, {K,E : 2}}	{K,E : 3}
M	{{K,E : 2}, {K : 1}}	{K : 3}
E	{K : 4}	{K : 4}
K		

- For each row, two types of association rules can be inferred for example for the first row which contains the element, the rules K -> Y and Y -> K can be inferred. To determine the valid rule, the confidence of both the rules is calculated and the one with confidence greater than or equal to the minimum confidence value is retained.

Construct FP tree from a Transaction DB

Steps for compressing the database into a FP tree:

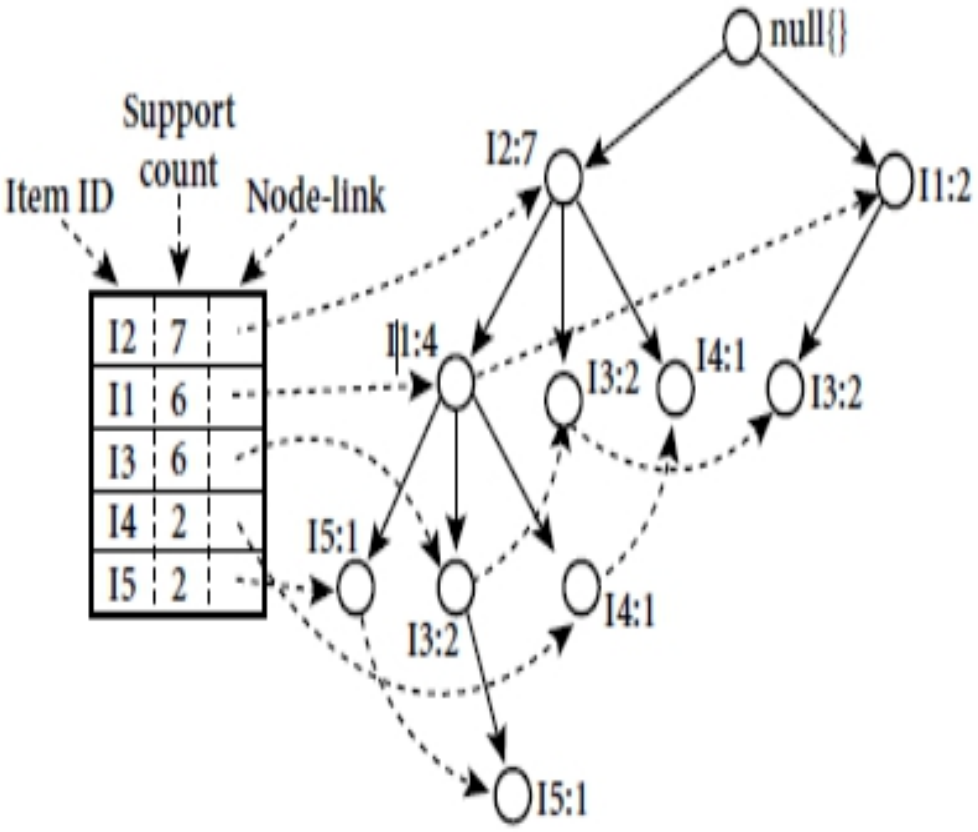
1. Scan DB once, find frequent 1 itemsets (single items)
2. Order frequent items in frequency descending order

Example1

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

Let's look at a concrete example, based on the *AllElectronics* transaction database,
D, There are nine transactions in this database, that is, $|D| = 9$. The Apriori algorithm is used for finding frequent itemsets in *D*.

An FP-tree registers compressed, frequent pattern information

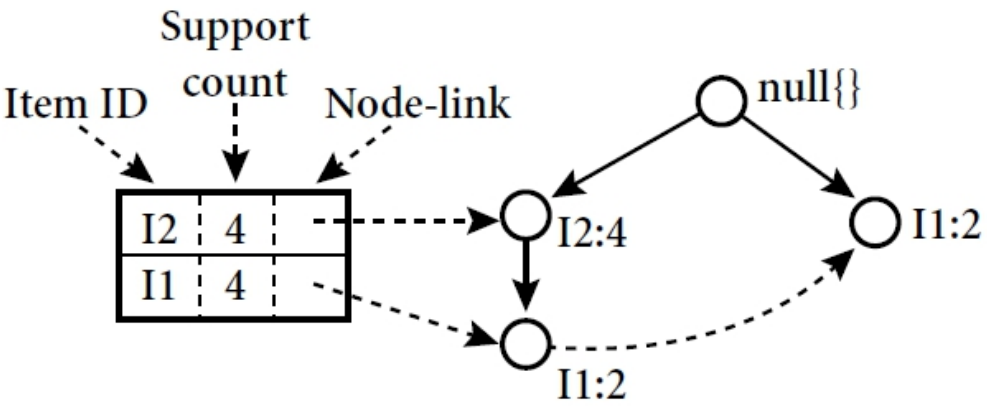


Contd.

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

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}

- **The conditional FP-tree associated with the conditional node I3**



How to Count Supports of Candidates?

- Why counting supports of candidates a problem?
  - The total number of candidates can be very huge
  - One transaction may contain many candidates
- Method:
  - Candidate itemsets are stored in a *hash-tree*
  - *Leaf* node of hash-tree contains a list of itemsets and counts
  - *Interior* node contains a hash table
  - *Subset function*: finds all the candidates contained in a transaction

Partition: Scan Database Only Twice

- Any itemset that is potentially frequent in DB must be frequent in at least one of the partitions of DB
  - Scan 1: partition database and find local frequent patterns
  - Scan 2: consolidate global frequent patterns
- A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association in large databases. In *VLDB'95*

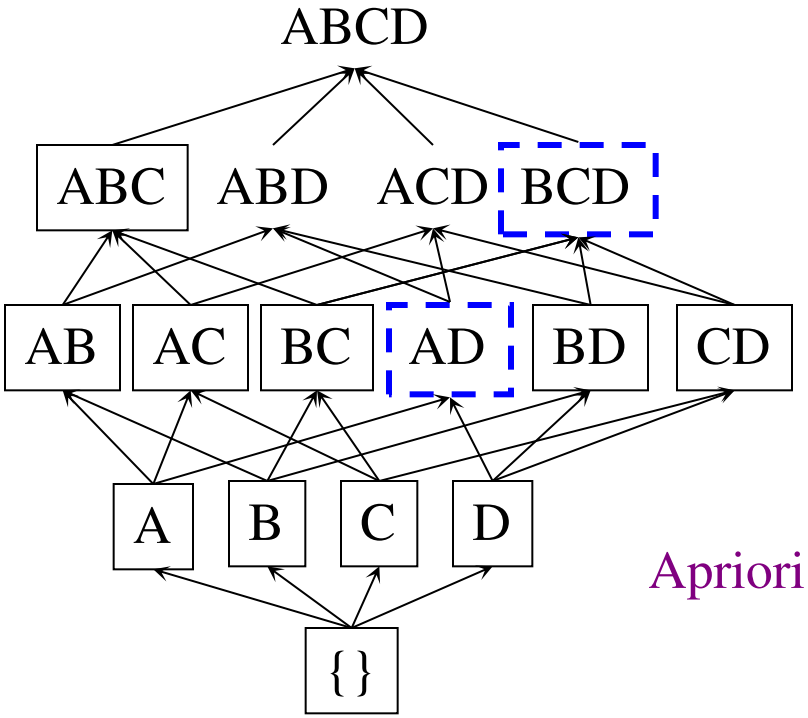
DHP: Reduce the Number of Candidates

- A *k*-itemset whose corresponding hashing bucket count is below the threshold cannot be frequent
  - Candidates: a, b, c, d, e
  - Hash entries: {ab, ad, ae} {bd, be, de} …
  - Frequent 1-itemset: a, b, d, e
  - ab is not a candidate 2-itemset if the sum of count of {ab, ad, ae} is below support threshold
- J. Park, M. Chen, and P. Yu. An effective hash-based algorithm for mining association rules. In *SIGMOD'95*

Sampling for Frequent Patterns

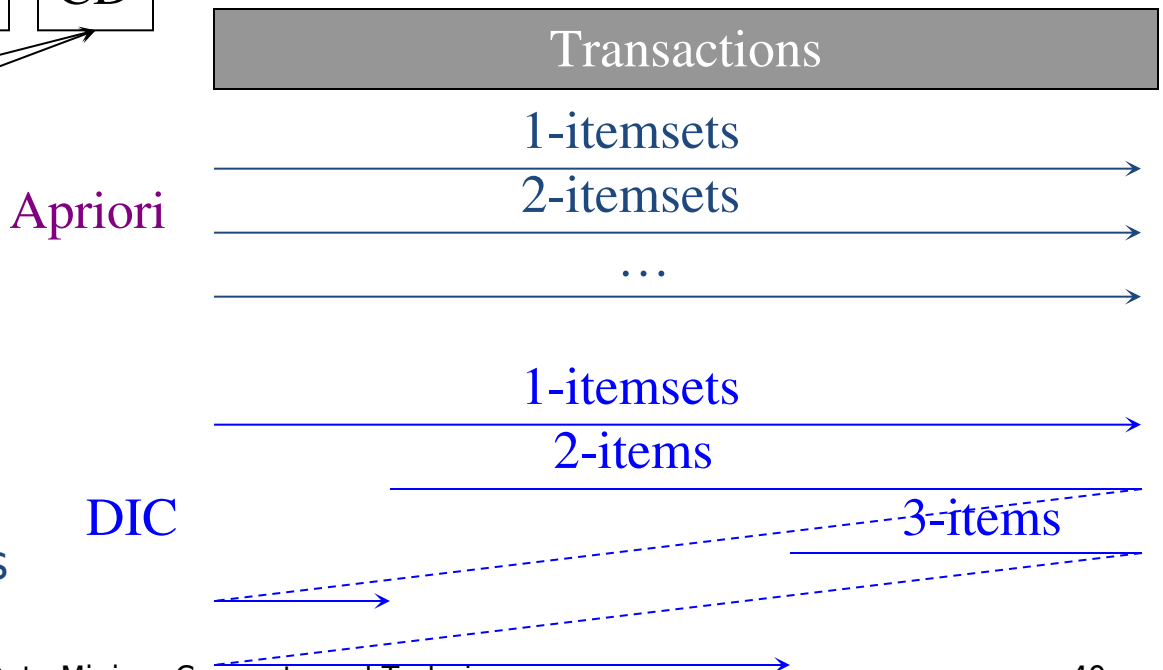
- Select a sample of original database, mine frequent patterns within sample using Apriori
- Scan database once to verify frequent itemsets found in sample, only *borders* of closure of frequent patterns are checked
  - Example: check *abcd* instead of *ab, ac, …, etc.*
- Scan database again to find missed frequent patterns
- H. Toivonen. Sampling large databases for association rules. In *VLDB'96*

DIC: Reduce Number of Scans



Itemset lattice

- Once both A and D are determined frequent, the counting of AD begins
- Once all length-2 subsets of BCD are determined frequent, the counting of BCD begins



S. Brin R. Motwani, J. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *SIGMOD'97*

Bottleneck of Frequent-pattern Mining

- Multiple database scans are costly
- Mining long patterns needs many passes of scanning and generates lots of candidates
  - To find frequent itemset $i_1 i_2 \ldots i_{100}$
    - # of scans: 100
    - # of Candidates: $\binom{100}{1} + \binom{100}{2} + \ldots + \binom{1}{1}\binom{0}{0}\binom{0}{0} = 2^{100} - 1 = 1.27 * 10^{30}$!
- Bottleneck: candidate-generation-and-test
- Can we avoid candidate generation?

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

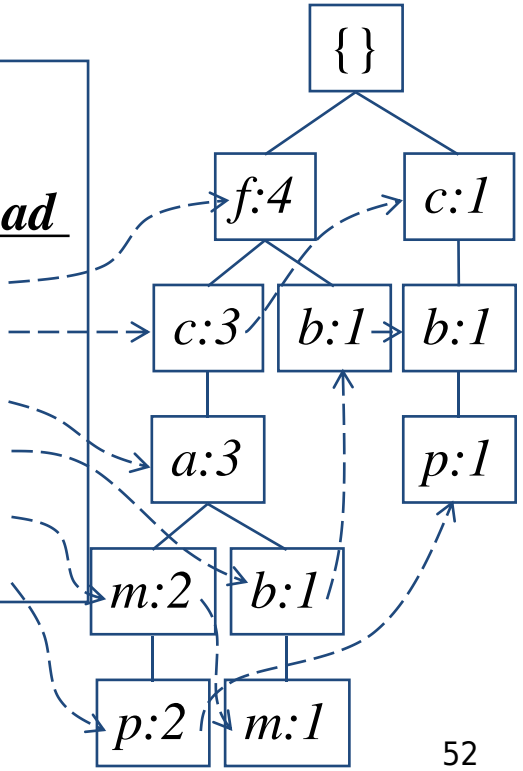
TID	Items bought	(ordered) frequent items
100	{f, a, c, d, g, i, m, p}	{f, c, a, m, p}
200	{a, b, c, f, l, m, o}	{f, c, a, b, m}
300	{b, f, h, j, o, w}	{f, b}
400	{b, c, k, s, p}	{c, b, p}
500	{a, f, c, e, l, p, m, n}	{f, c, a, m, p}

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

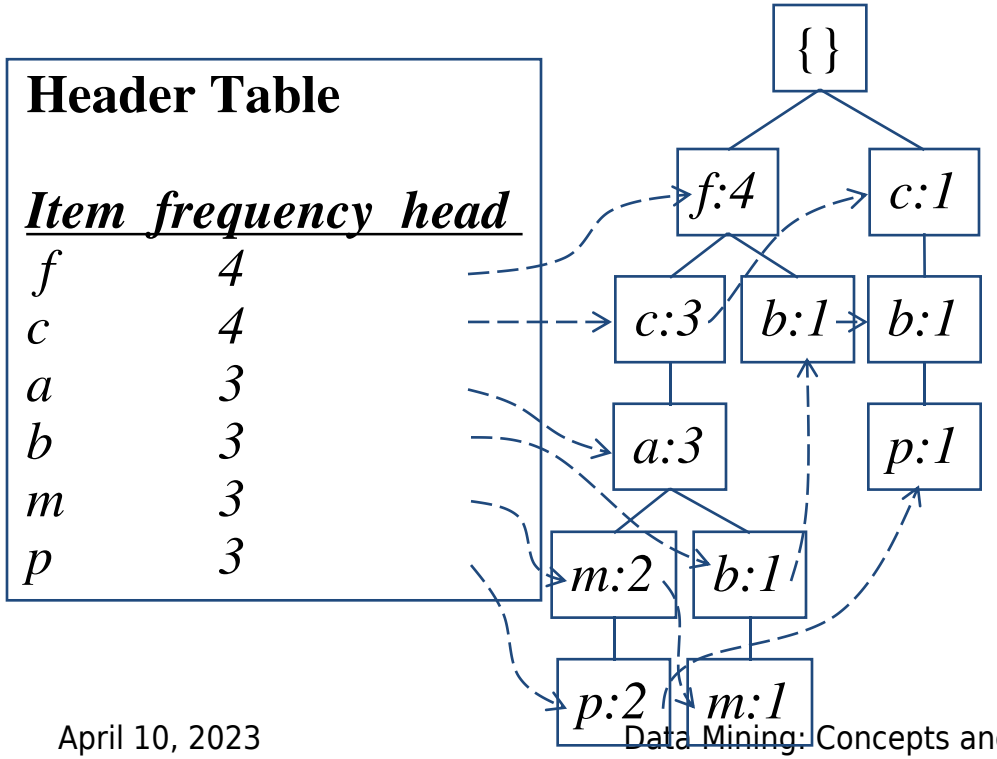
- Completeness
  - Preserve complete information for frequent pattern mining
  - Never break a long pattern of any transaction
- Compactness
  - Reduce irrelevant info—infrequent items are gone
  - Items in frequency descending order: the more frequently occurring, the more likely to be shared
  - Never be larger than the original database (not count node-links and the *count* field)
  - For Connect-4 DB, compression ratio could be over 100

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

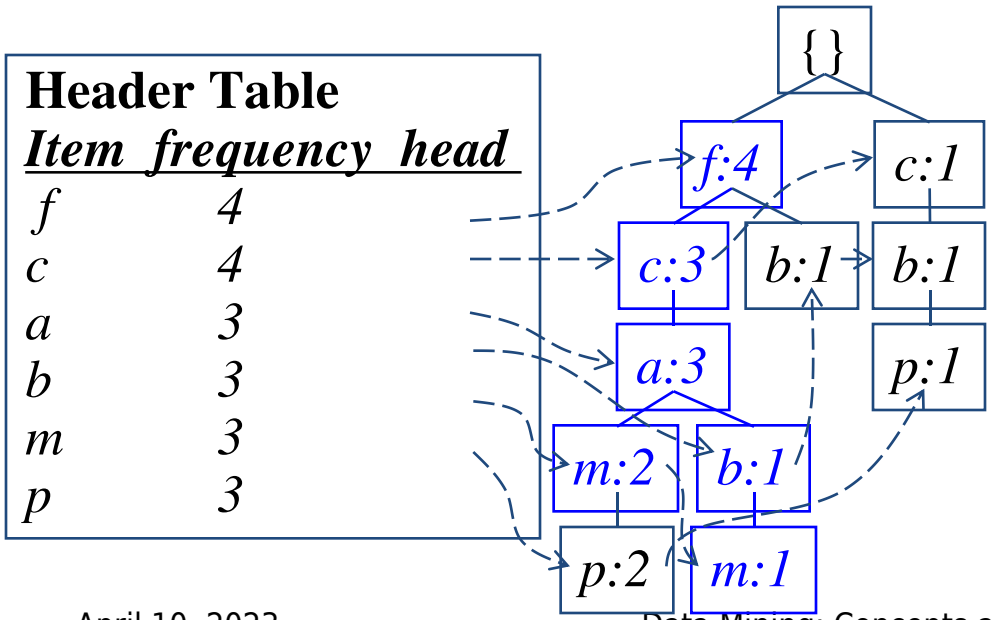


***Conditional* pattern bases**

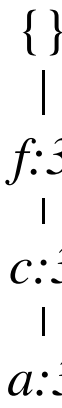
item	*cond. pattern base*
c	***f:3***
a	***fc:3***
b	***fca:1, f:1, c:1***
m	***fca:2, fcab:1***
p	***fcam:2, cb:1***

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



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



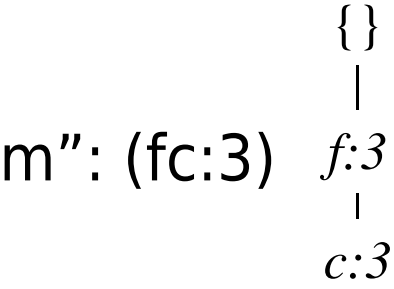
***m-conditional* FP-tree**

All frequent patterns relate to *m*

m,
fm, cm, am,
fcm, fam, cam,
fcam

Recursion: Mining Each Conditional FP-tree

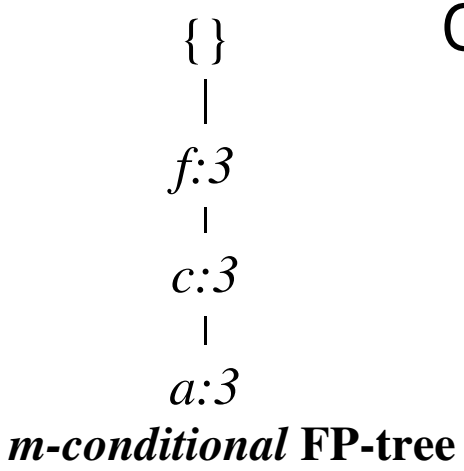
Cond. pattern base of “am”: (fc:3)



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

Cond. pattern base of “cm”: (f:3)

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



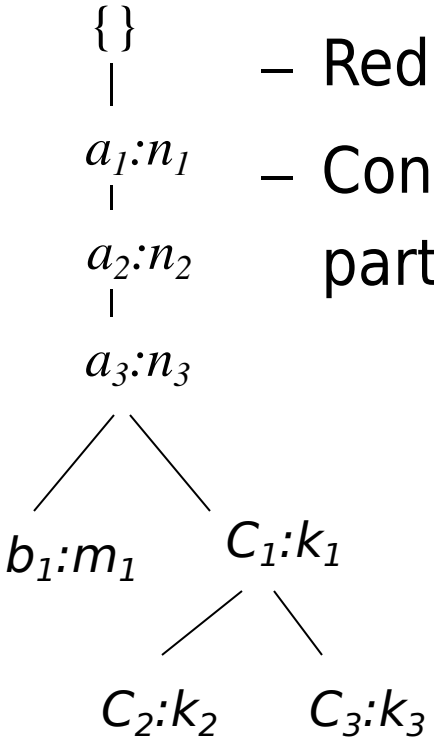
***m-conditional* FP-tree**

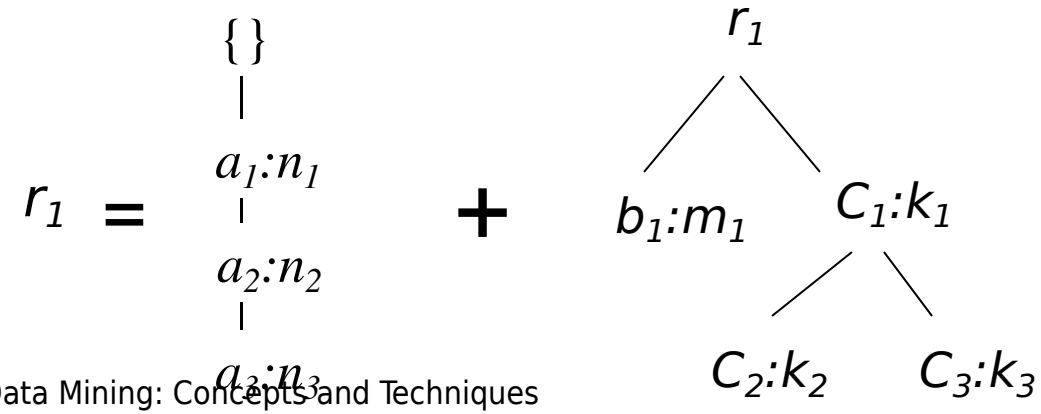
Cond. pattern base of “cam”: (f:3)

***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





Mining Frequent Patterns With FP-trees

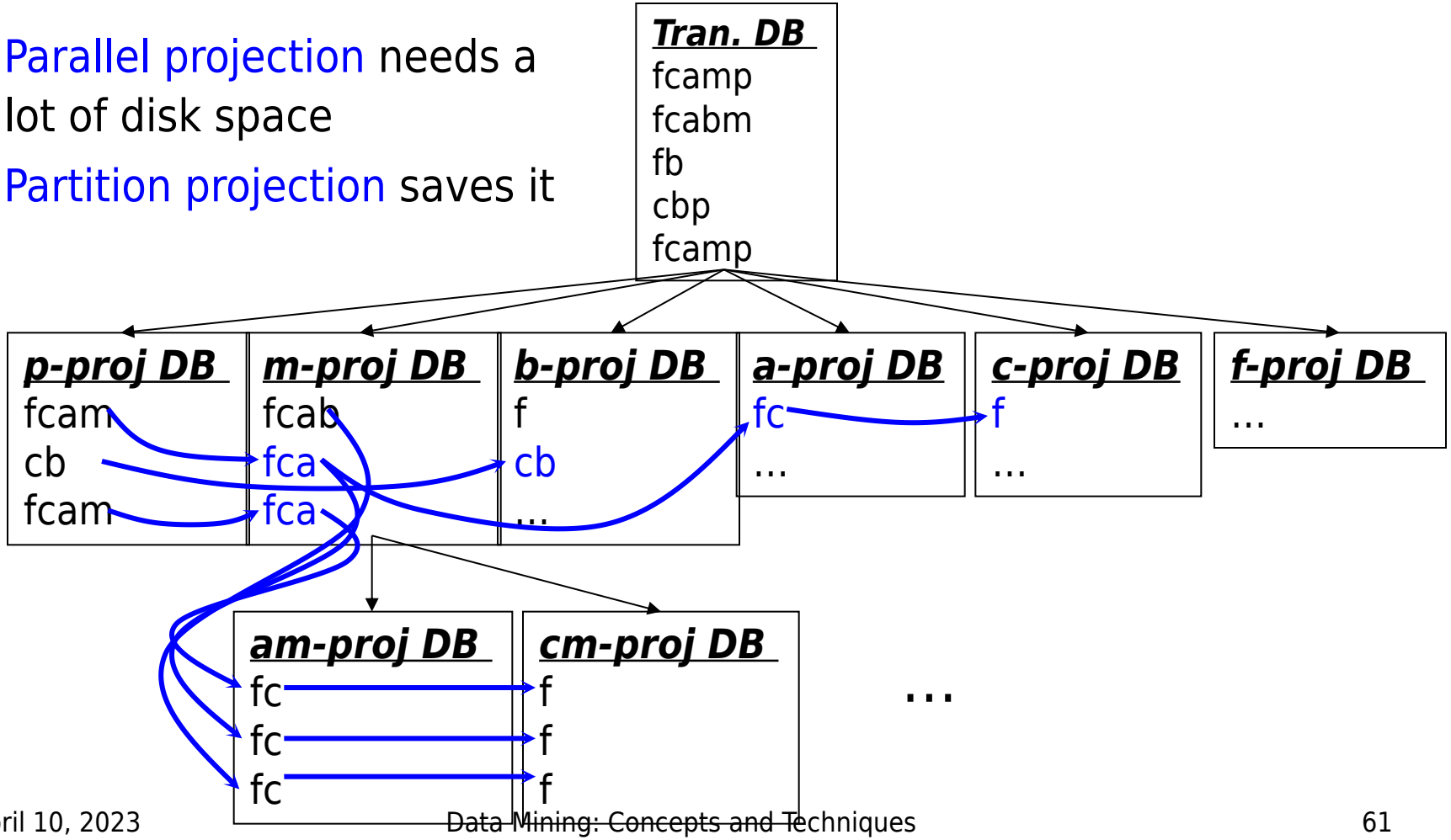
- Idea: Frequent pattern growth
  - Recursively grow frequent patterns by pattern and database partition
- Method
  - For each frequent item, construct its conditional pattern-base, and then its conditional FP-tree
  - Repeat the process on each newly created conditional FP-tree
  - Until the resulting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

Scaling FP-growth by DB Projection

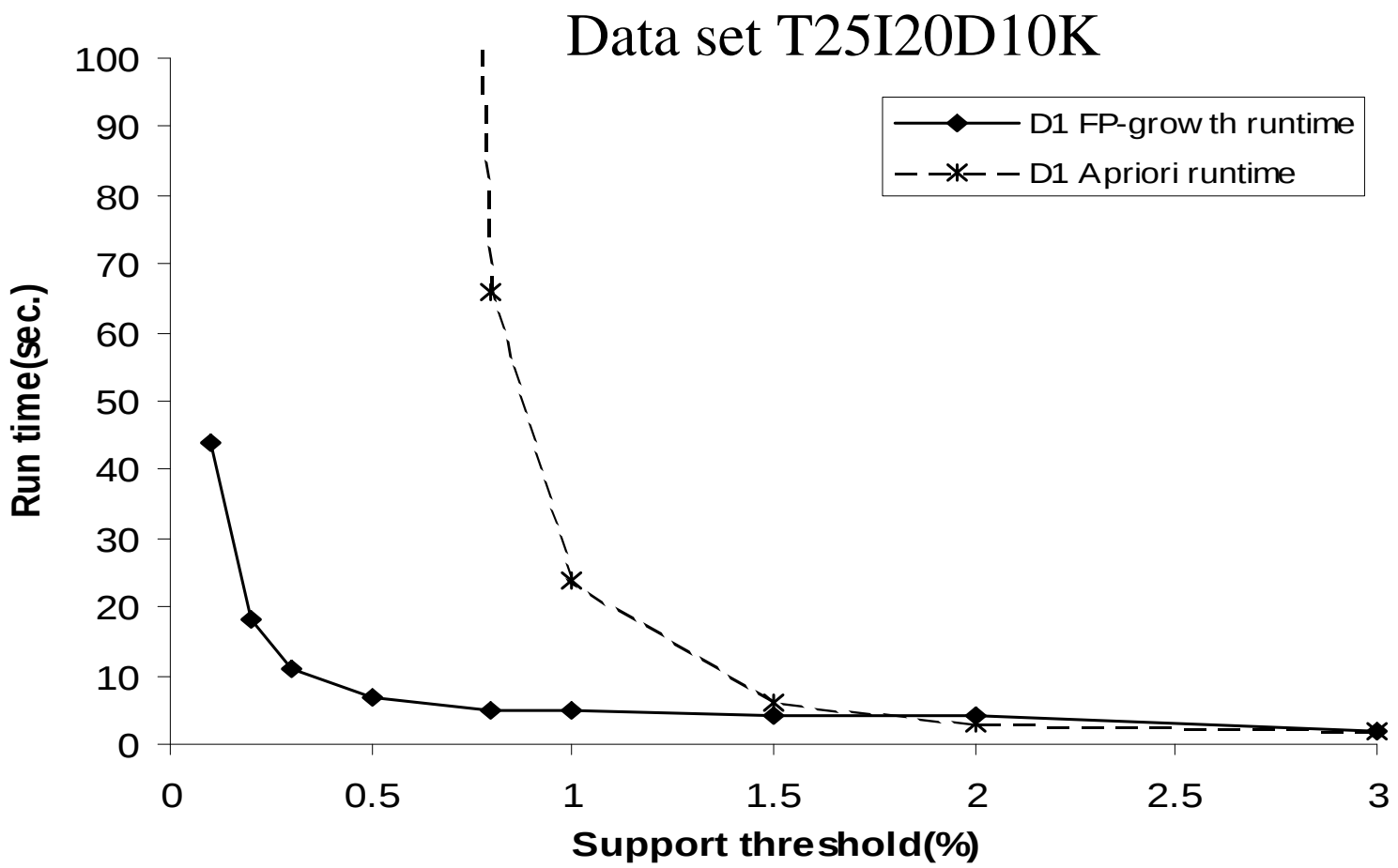
- FP-tree cannot fit in memory?—DB projection
- First partition a database into a set of projected DBs
- Then construct and mine FP-tree for each projected DB
- Parallel projection vs. Partition projection techniques
  - Parallel projection is space costly

Partition-based Projection

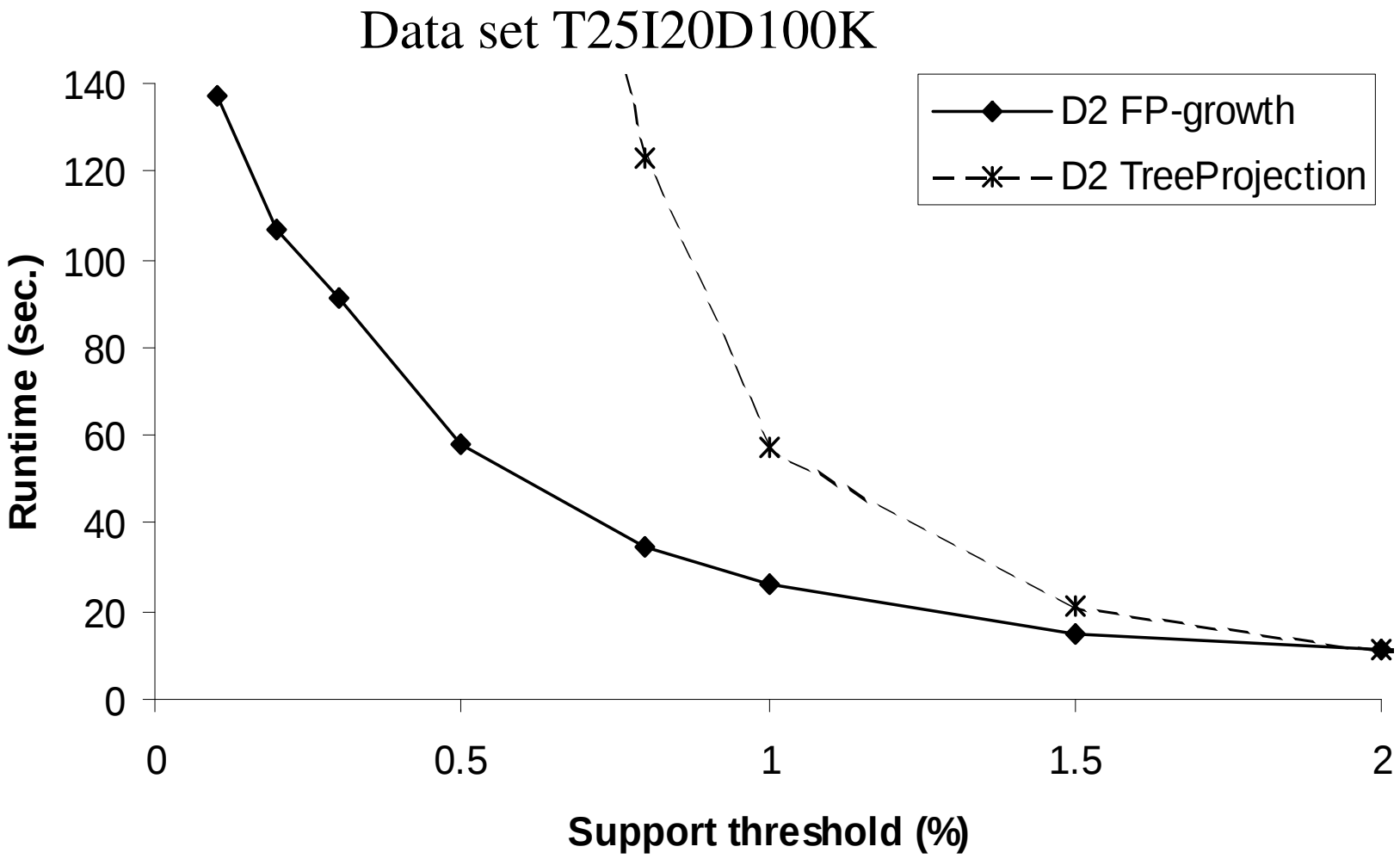
- Parallel projection needs a lot of disk space
- Partition projection saves it



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



FP-Growth vs. Tree-Projection: Scalability with the Support Threshold



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

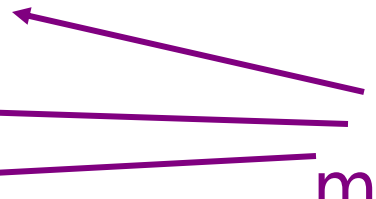
Implications of the Methodology

- Mining closed frequent itemsets and max-patterns
  - CLOSET (DMKD’00)
- Mining sequential patterns
  - FreeSpan (KDD’00), PrefixSpan (ICDE’01)
- Constraint-based mining of frequent patterns
  - Convertible constraints (KDD’00, ICDE’01)
- Computing iceberg data cubes with complex measures
  - H-tree and H-cubing algorithm (SIGMOD’01)

MaxMiner: Mining Max-patterns

- 1st scan: find frequent items
  - A, B, C, D, E
- 2nd scan: find support for
  - AB, AC, AD, AE, ABCDE
  - BC, BD, BE, BCDE
  - CD, CE, CDE, DE,

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



- Since BCDE is a max-pattern, no need to check BCD, BDE, CDE in later scan
- R. Bayardo. Efficiently mining long patterns from databases. In *SIGMOD'98*

Mining Frequent Closed Patterns: CLOSET

- Flist: list of all frequent items in support ascending order
  - Flist: d-a-f-e-c
- Divide search space
  - Patterns having d
  - Patterns having d but no a, etc.
- Find frequent closed pattern recursively
  - Every transaction having d also has cfa cfad is a frequent closed pattern
- J. Pei, J. Han & R. Mao. CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets", DMKD'00.

Min_sup=2

TID	Items
10	a, c, d, e, f
20	a, b, e
30	c, e, f
40	a, c, d, f
50	c, e, f

CLOSET+: Mining Closed Itemsets by Pattern-Growth

- Itemset merging: if Y appears in every occurrence of X, then Y is merged with X
- Sub-itemset pruning: if Y ↄ X, and sup(X) = sup(Y), X and all of X's descendants in the set enumeration tree can be pruned
- Hybrid tree projection
  - Bottom-up physical tree-projection
  - Top-down pseudo tree-projection
- Item skipping: if a local frequent item has the same support in several header tables at different levels, one can prune it from the header table at higher levels
- Efficient subset checking

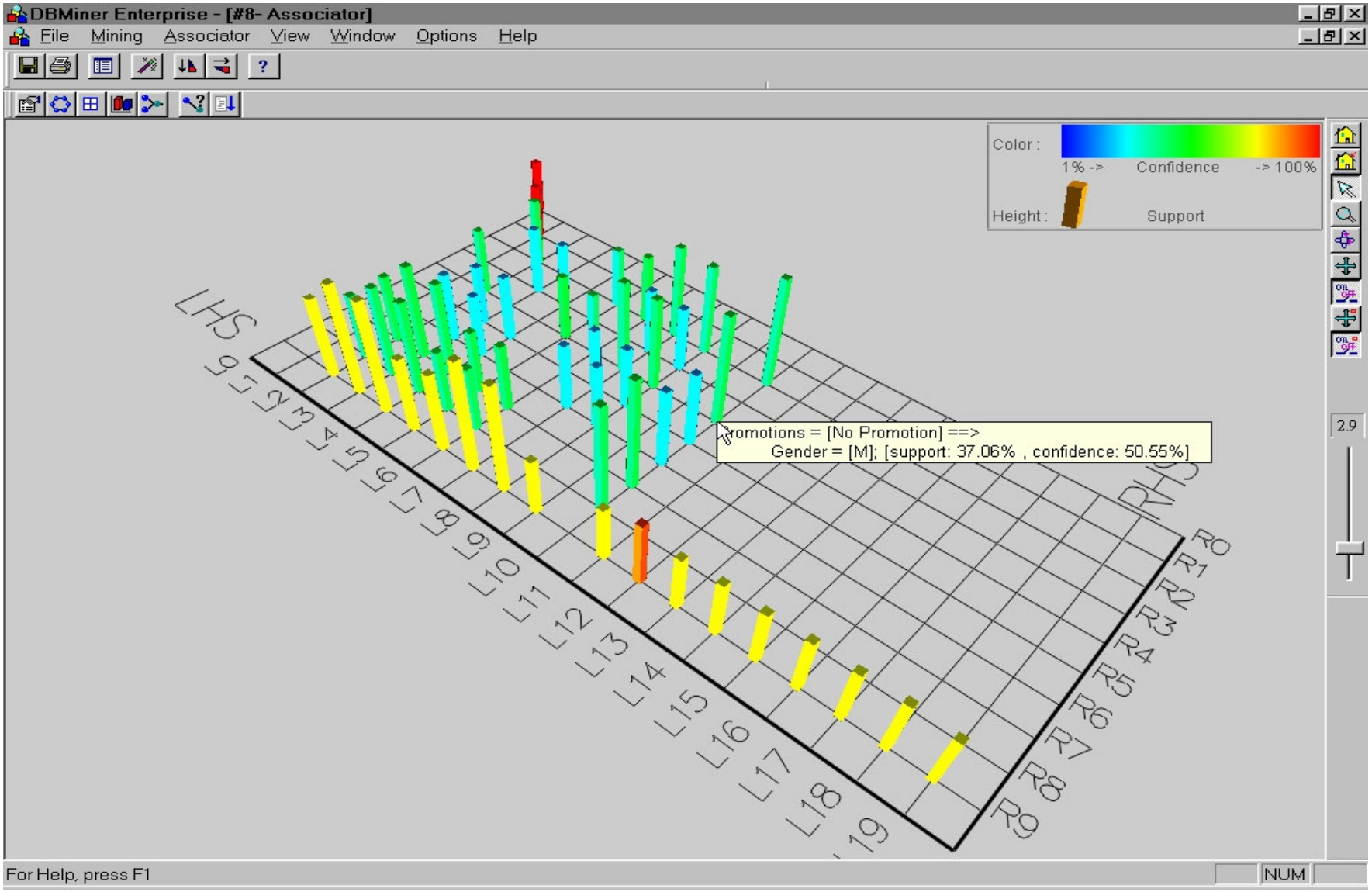
CHARM: Mining by Exploring Vertical Data Format

- Vertical format: t(AB) = $\{T_{11}, T_{25}, \ldots\}$
  - tid-list: list of trans.-ids containing an itemset
- Deriving closed 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
- Using diffset to accelerate mining
  - Only keep track of differences of tids
  - t(X) = $\{T_1, T_2, T_3\}$, t(XY) = $\{T_1, T_3\}$
  - Diffset (XY, X) = $\{T_2\}$
- Eclat/MaxEclat (Zaki et al. @KDD'97), VIPER(P. Shenoy et al.@SIGMOD'00), CHARM (Zaki & Hsiao@SDM'02)

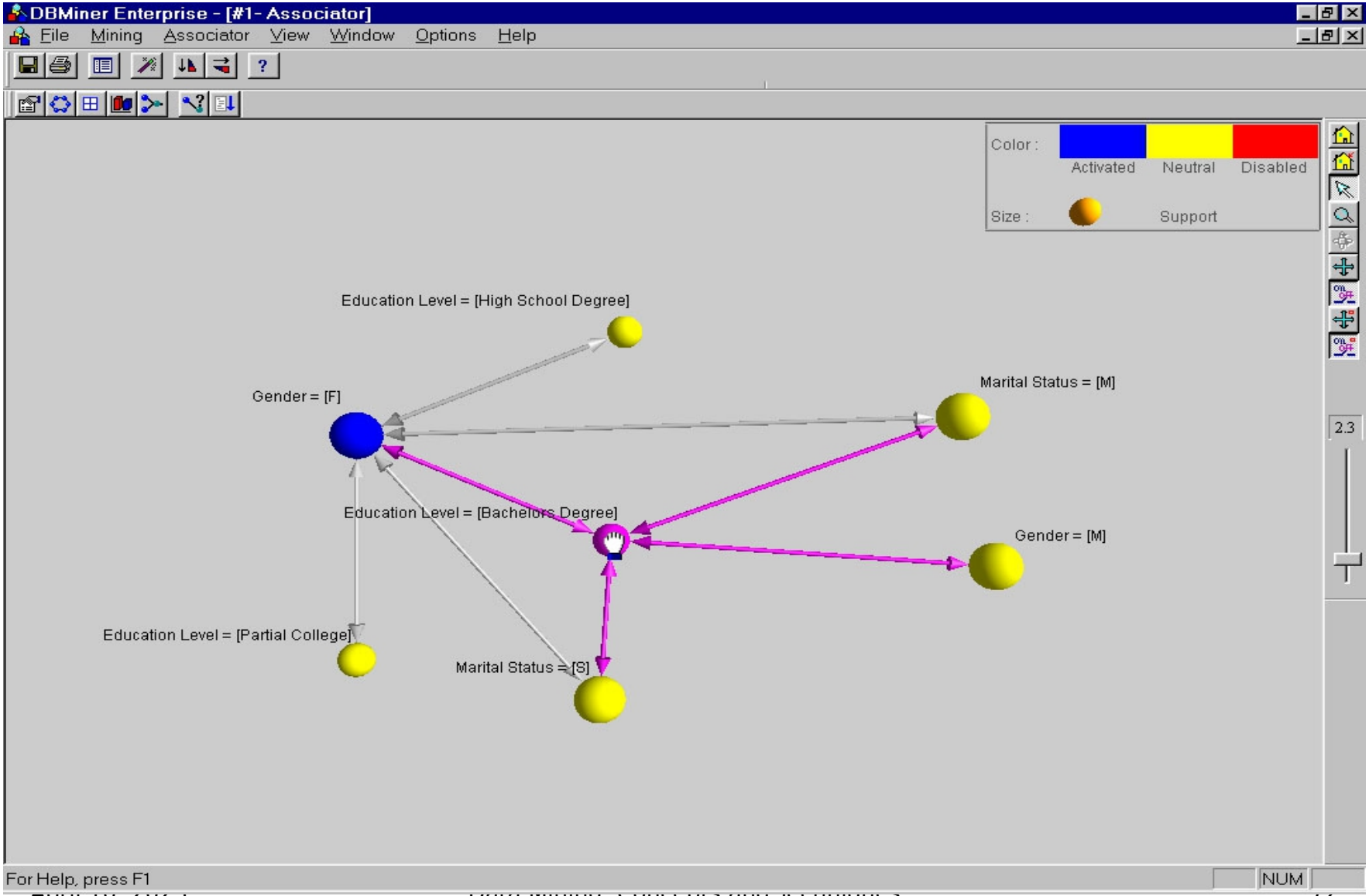
Further Improvements of Mining Methods

- AFOPT (Liu, et al. @ KDD’03)
  - A “push-right” method for mining condensed frequent pattern (CFP) tree
- Carpenter (Pan, et al. @ KDD’03)
  - Mine data sets with small rows but numerous columns
  - Construct a row-enumeration tree for efficient mining

Visualization of Association Rules: Plane Graph



Visualization of Association Rules: Rule Graph



Visualization of Association Rules (SGI/MineSet 3.0)