Unit II : ASSOCIATION RULES

Basic Concepts - Market Basket Analysis - Frequent Itemsets, Closed Itemsets and Association Rules - Frequent Itemset Mining Methods – Apriori Algorithm – Generating Association Rules - **Frequent pattern growth** - Mining Various Kinds of Association Rules

Frequent Patterns

- Frequent pattern: a pattern (a set of items, subsequences, substructures, etc.) that occurs frequently in a data set
  - itemset: A set of one or more items
  - k-itemset: $X = \{x_1, \ldots, x_k\}$
  - Mining algorithms
    - Apriori
    - FP-growth

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

Support & Confidence

- Support
  - (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
- Confidence (association rule: X→Y)
  - $sup(X \cup Y)/sup(x)$ (conditional prob.: $Pr(Y|X) = Pr(X \wedge Y)/Pr(X)$)
  - confidence, c, conditional probability that a transaction having X also contains Y
  - Find all the rules X→Y with minimum support and confidence
    - $sup(X \cup Y) \geq minsup$
    - $sup(X \cup Y)/sup(X) \geq minconf$

Algorithms to find frequent pattern

- **Apriori**: uses a generate-and-test approach – generates candidate itemsets and tests if they are frequent
  - Generation of candidate itemsets is expensive (in both space and time)
  - Support counting is expensive
    - Subset checking (computationally expensive)
    - Multiple Database scans (I/O)
- **FP-Growth**: allows frequent itemset discovery without candidate generation. Two step:
  - 1.Build a compact data structure called the FP-tree
    - 2 passes over the database
  - 2.extracts frequent itemsets directly from the FP-tree
    - Traverse through FP-tree

Pattern-Growth Approach: Mining Frequent Patterns Without Candidate Generation

- The FP-Growth Approach
  - Depth-first search (Apriori: Breadth-first search)
  - Avoid explicit candidate generation

FP-Growth approach:

- 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-tree construction:

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

Step 1: FP-Tree Construction

- FP-Tree is constructed using 2 passes over the data-set:

Pass 1:

- Scan data and find support for each item.
- Discard infrequent items.
- Sort frequent items in decreasing order based on their support.

Use this order when building the FP-Tree, so common prefixes can be shared.

Step 1: FP-Tree Construction

Pass 2:

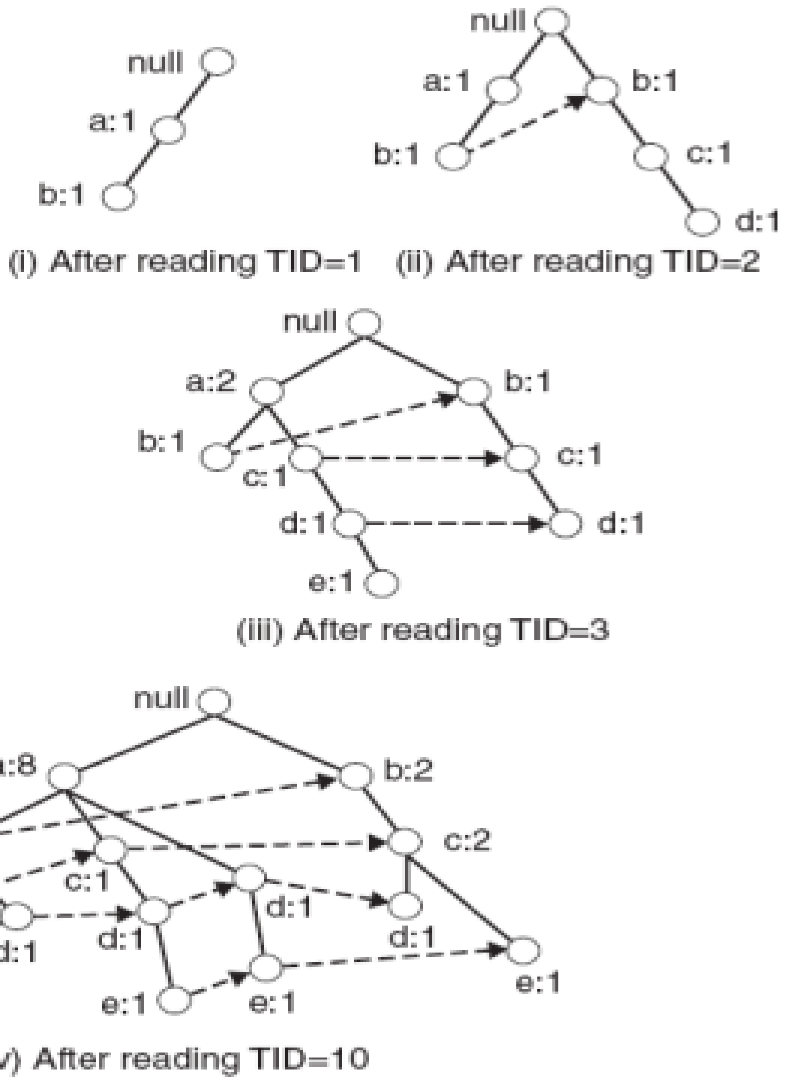
Nodes correspond to items and have a counter

1. FP-Growth reads 1 transaction at a time and maps it to a path
2. Fixed order is used, so paths can overlap when transactions share items (when they have the same prfix).
   - In this case, counters are incremented
3. Pointers are maintained between nodes containing the same item, creating singly linked lists (dotted lines)
   - The more paths that overlap, the higher the compression. FP-tree may fit in memory.
4. Frequent itemsets extracted from the FP-Tree.

Step 1: FP-Tree Construction (Example)

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 Size

- The size of an FP-tree is typically smaller than the size of the uncompressed data because many transactions often share a few items in common
  - **Best case scenario:** All transactions have the same set of items, and the FP-tree contains only a single branch of nodes.
  - **Worst case scenario:** Every transaction 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 original data.
- The size of an FP-tree also depends on how the items are ordered

Mining Frequent Patterns Without Candidate Generation

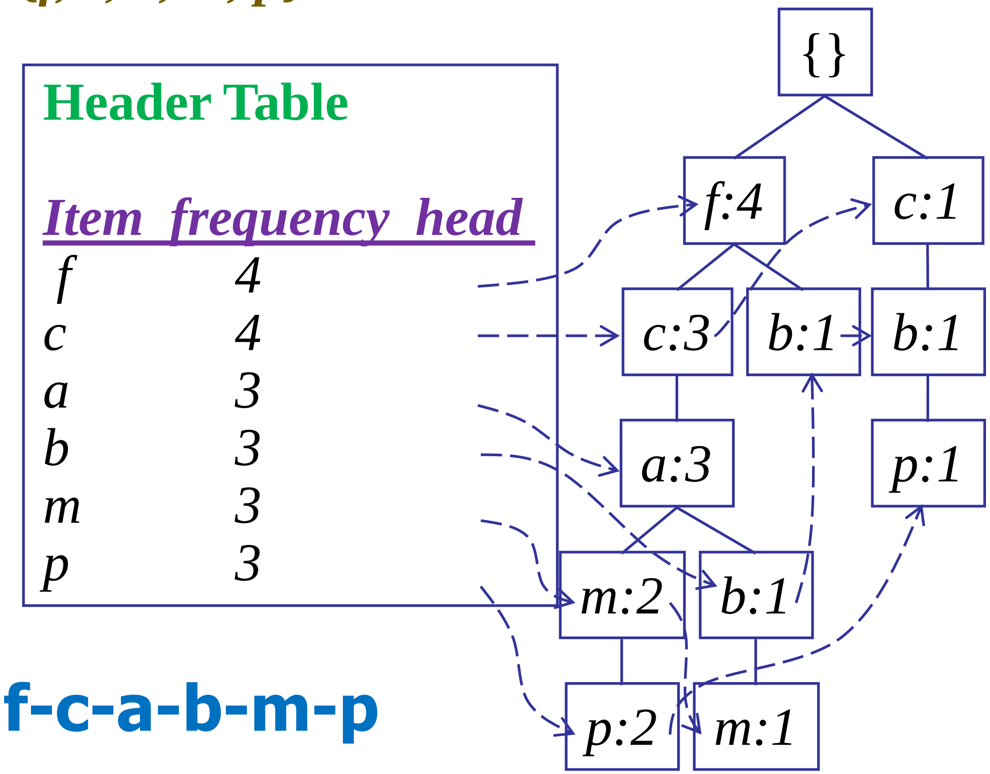
- Grow long patterns from short ones using local frequent items
  - "abc" is a frequent pattern
  - Get all transactions having "abc": DB|abc
  - "d" is a local frequent item in DB|abc → abcd is a frequent pattern

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



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

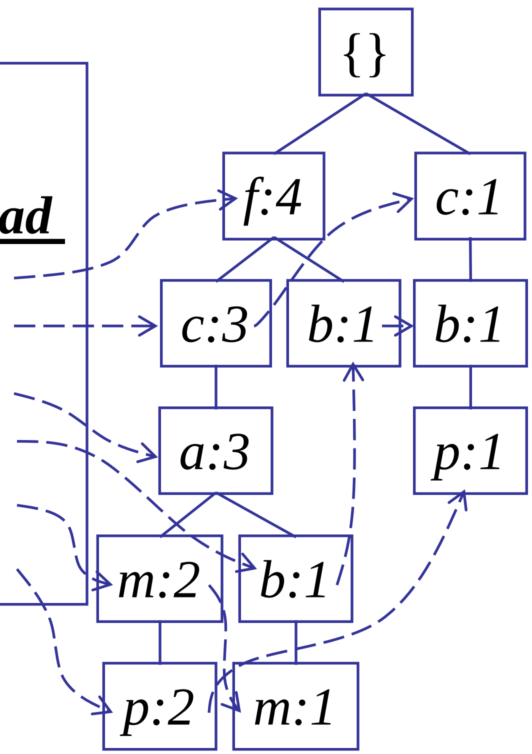
- Frequent patterns can be partitioned into subsets according to f-list
  - F-list=f-c-a-b-m-p
  - Patterns containing p
  - Patterns having m but no p
  - …
  - Patterns having c but no a nor b, m, p
  - Pattern f
- Completeness and non-redundency

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

Header Table

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

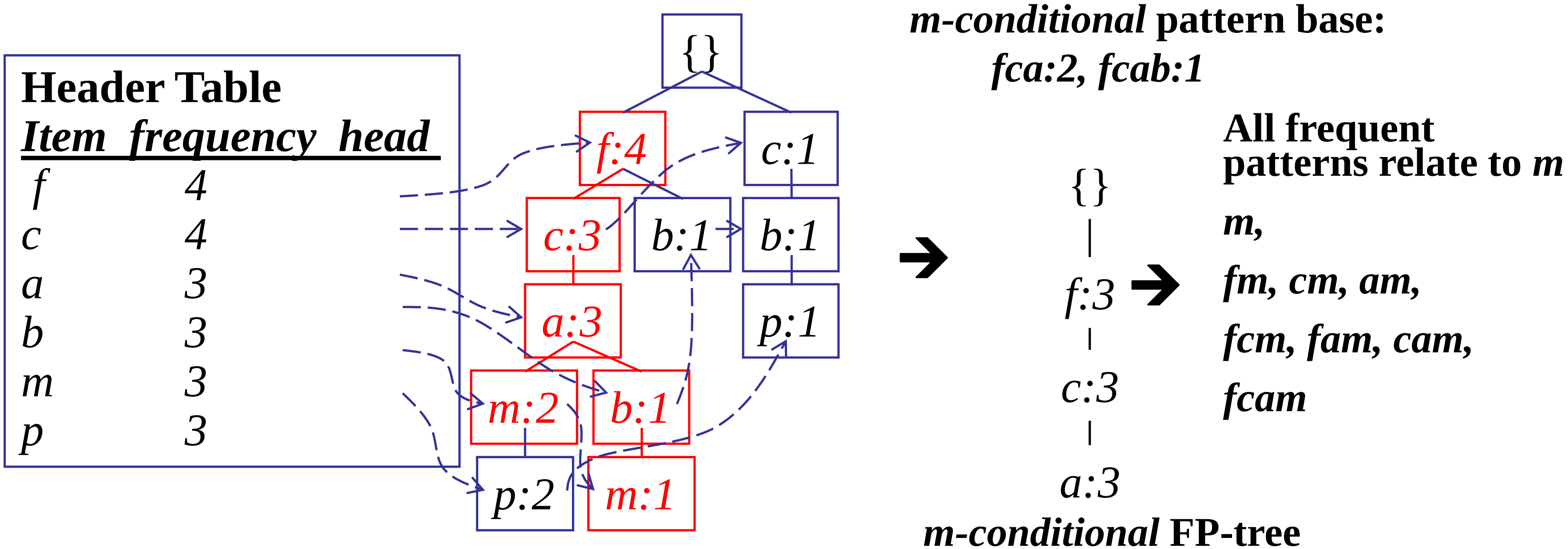


***Conditional* pattern bases**

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



Recursion: Mining Each Conditional FP-tree

{}
|
f:3
|
c:3
|
a:3
m-conditional **FP-tree**

Cond. pattern base of “am”: (fc:3)

{}
|
f:3
|
c:3
am-conditional **FP-tree**

Cond. pattern base of “cm”: (f:3)

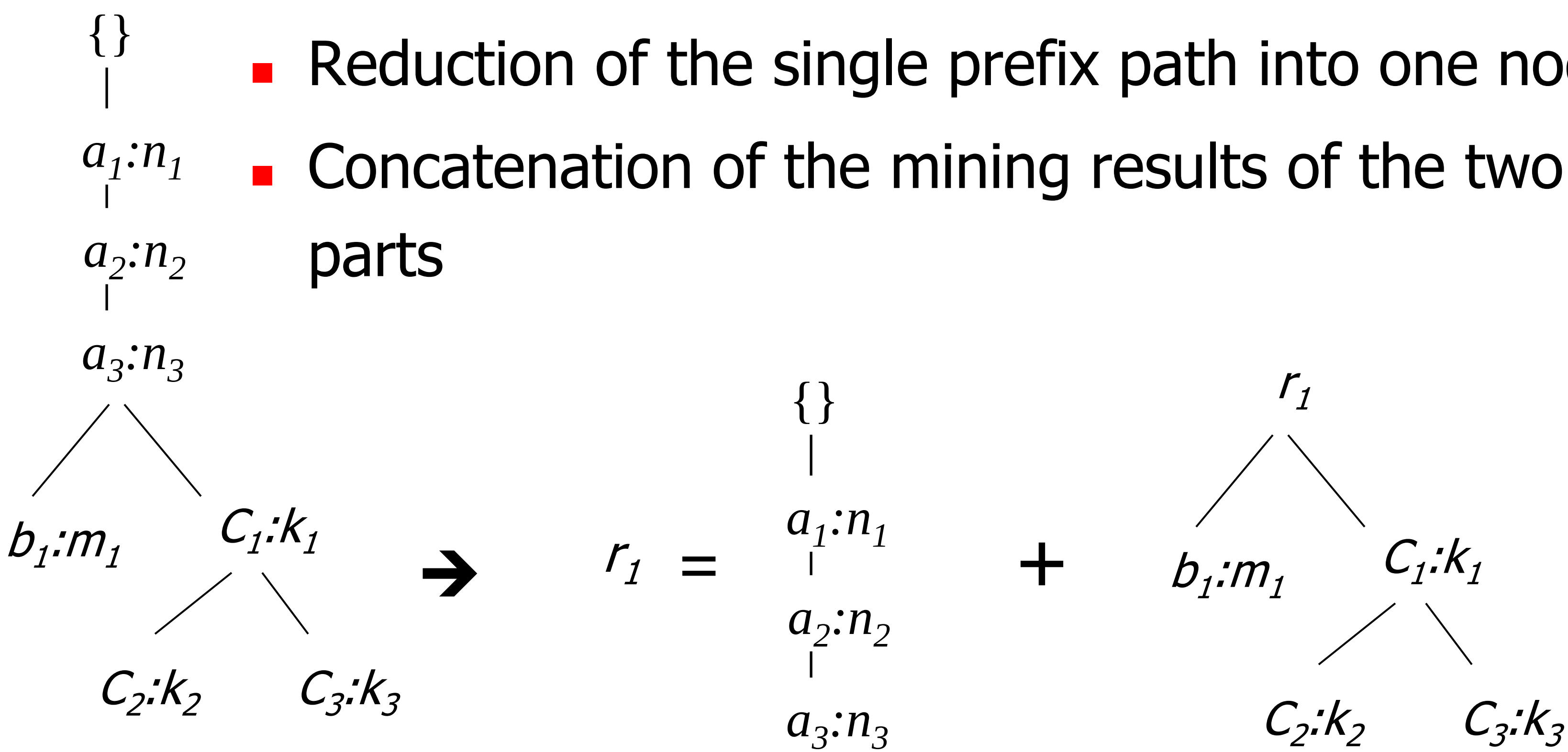
{}
|
f:3
cm-conditional **FP-tree**

Cond. pattern base of “cam”: (f:3)

{}
|
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

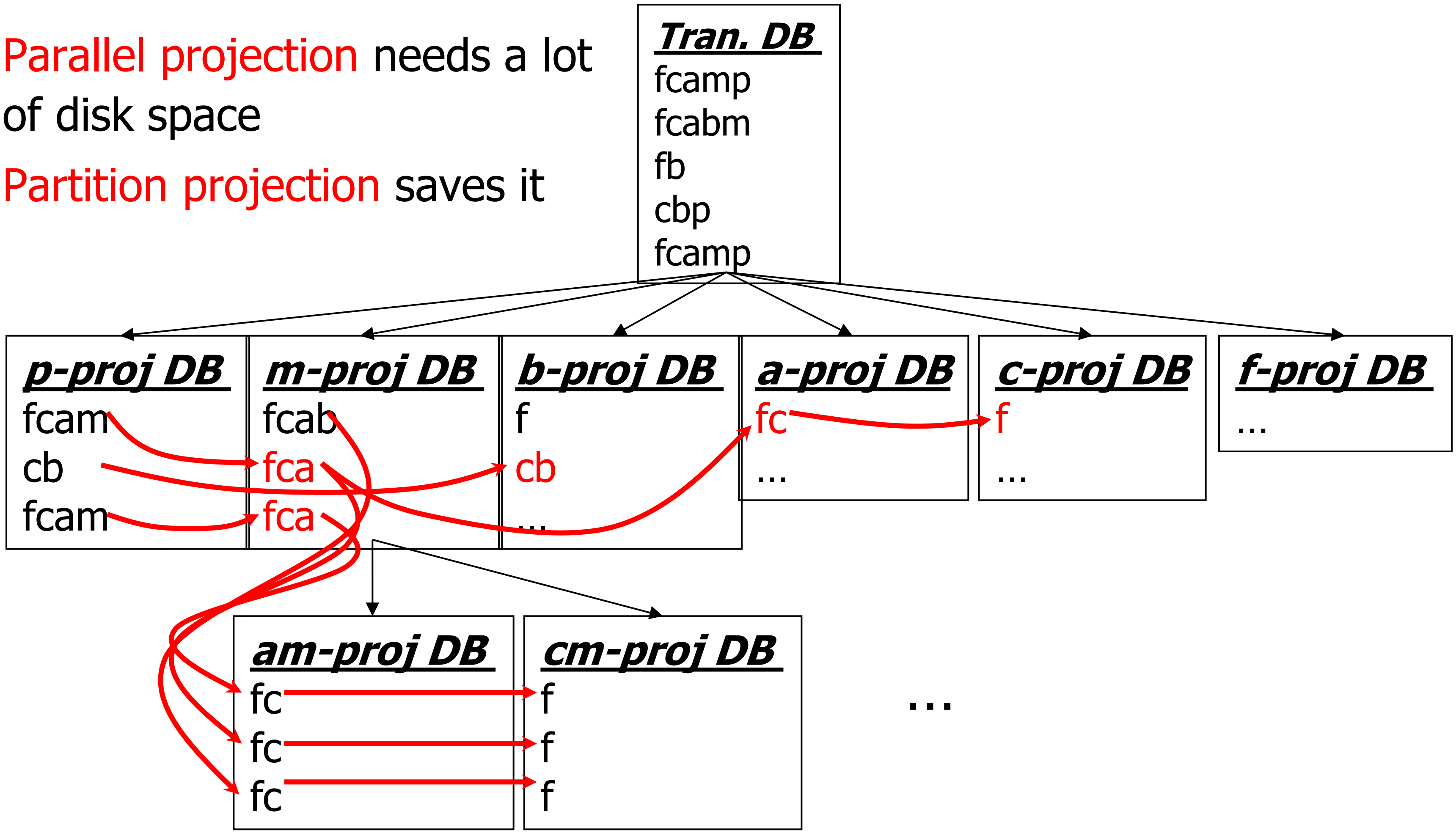
- Idea: Frequent pattern growth
  - Recursively grow frequent patterns by pattern and database partition
- Method
  - For each frequent item, construct its conditional pattern-base, and then its conditional FP-tree
  - Repeat the process on each newly created conditional FP-tree
  - Until the resulting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

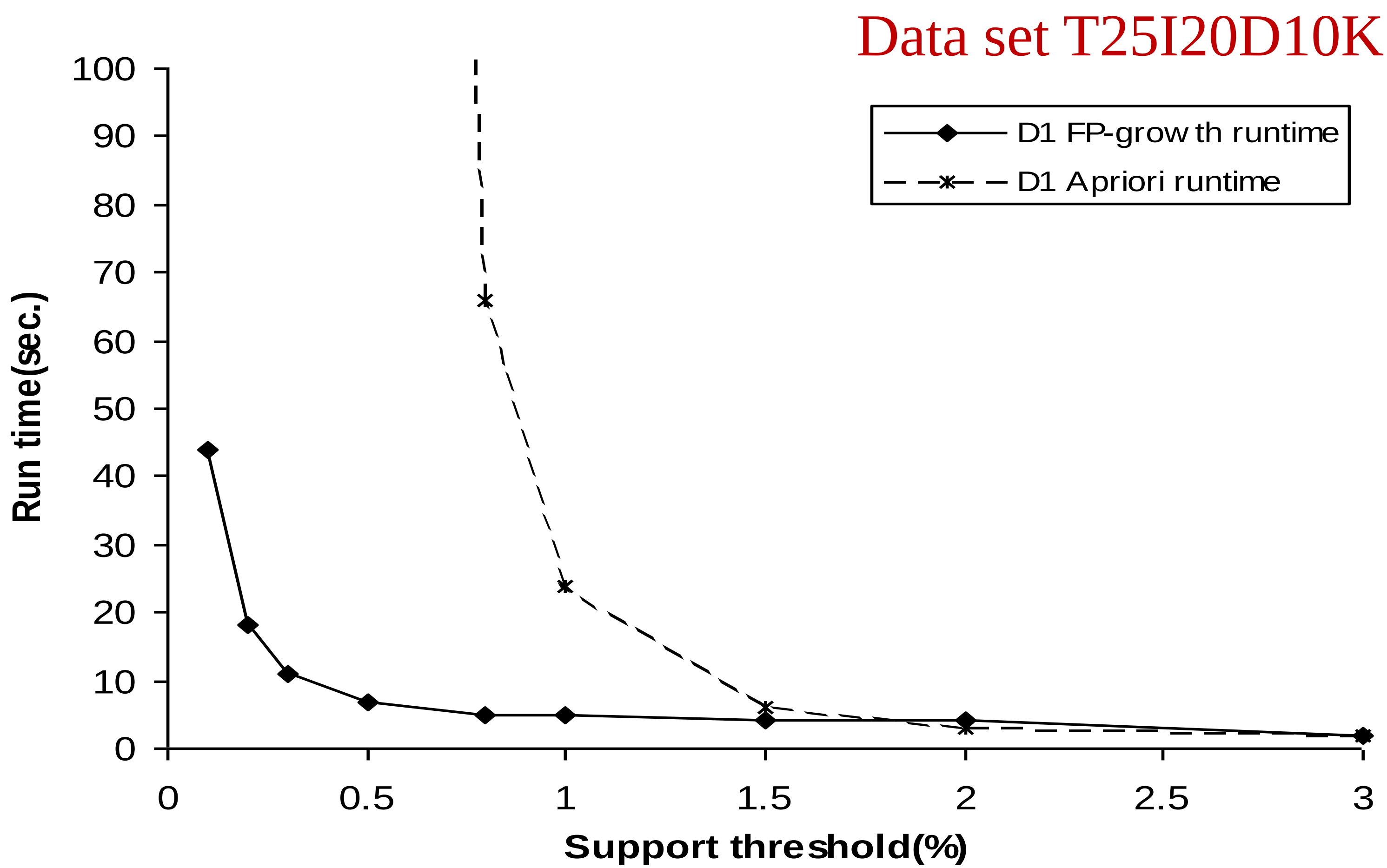
- FP-tree cannot fit in memory?—DB projection
- First partition a database into a set of projected DBs
- Then construct and mine FP-tree for each projected DB
- Parallel projection vs. Partition projection techniques
  - Parallel projection is space costly

Partition-based Projection

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

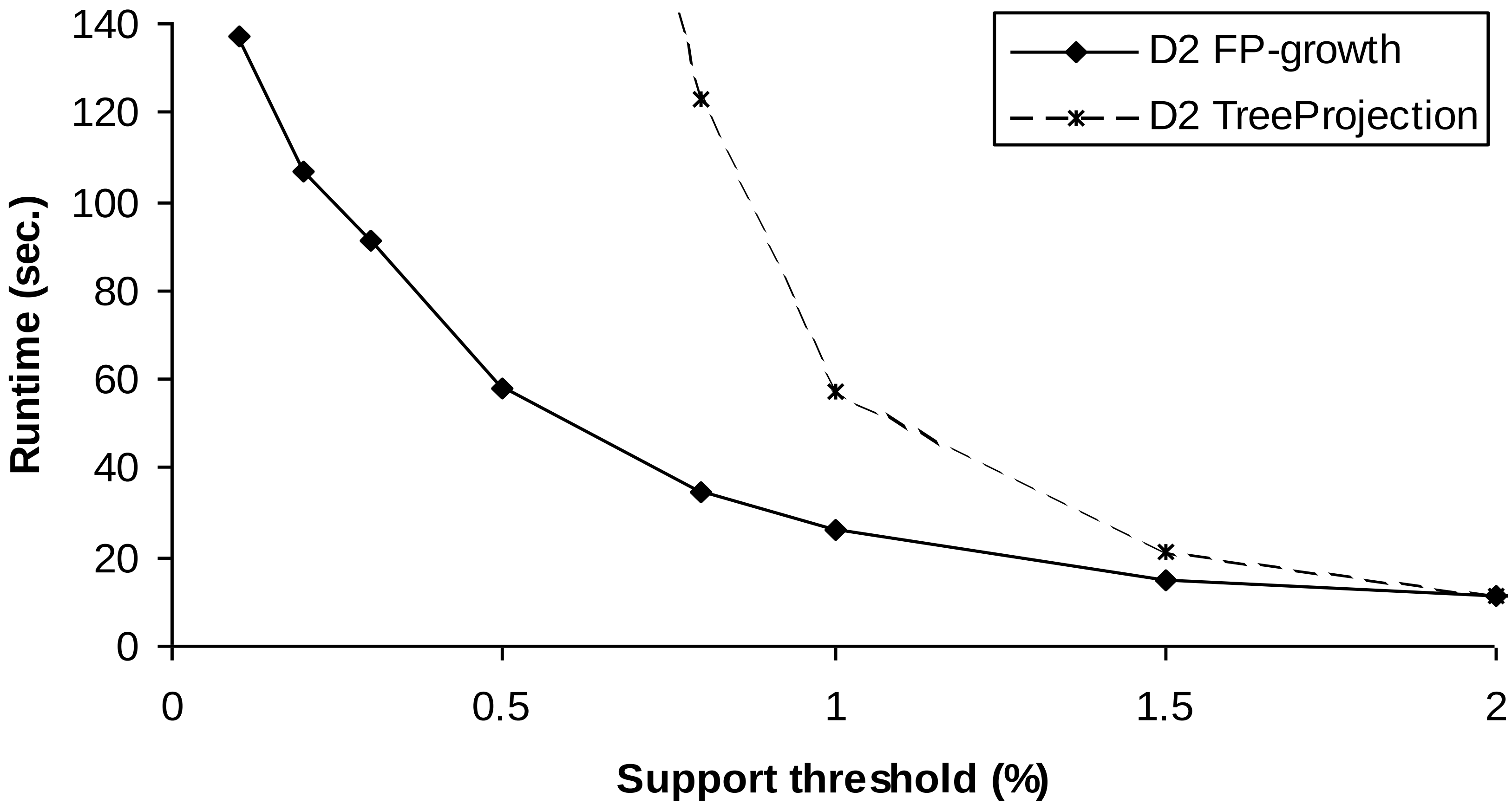


FP-Growth vs. Apriori: Scalability With the Support Threshold



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

Data set T25I20D100K



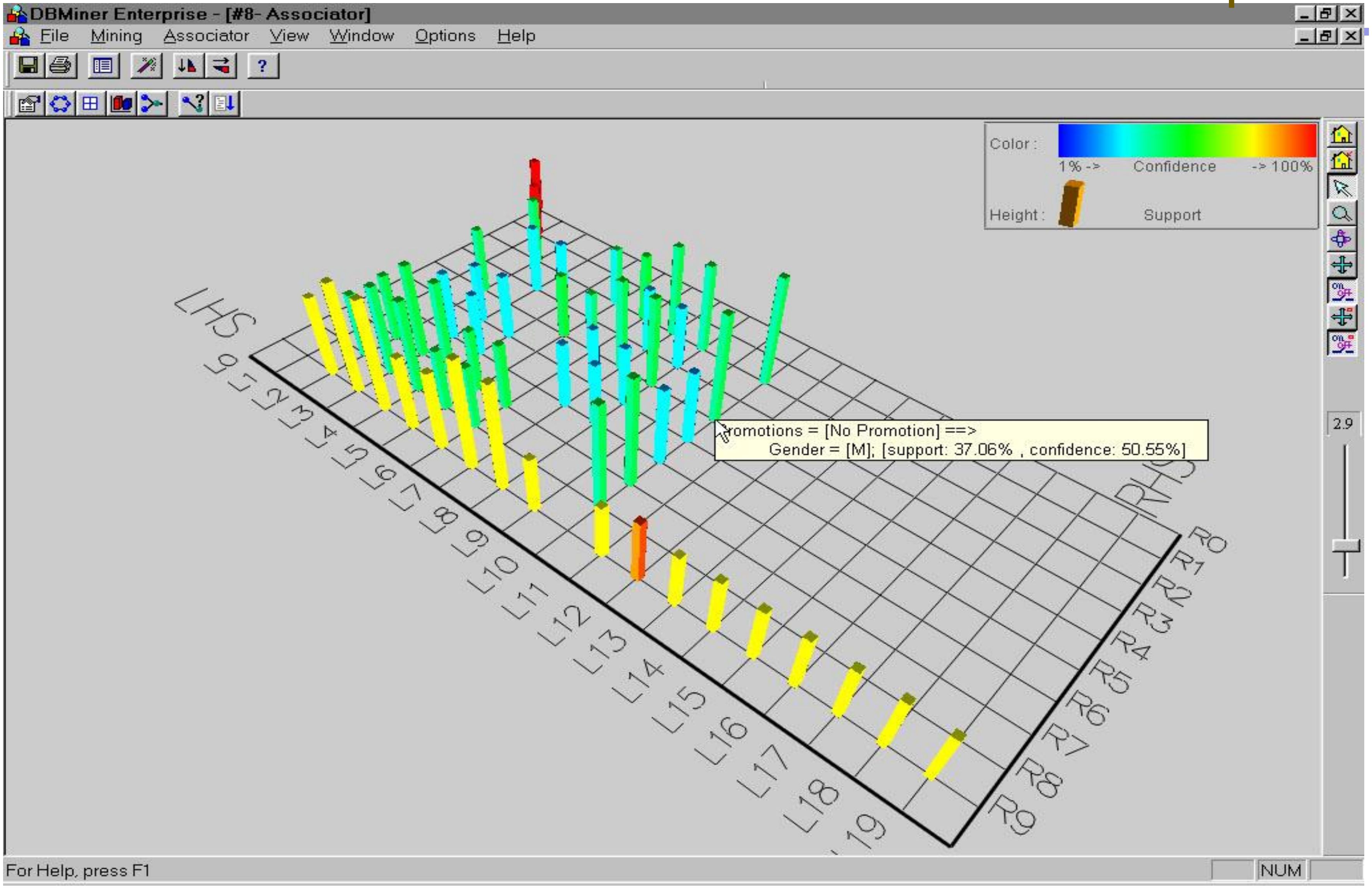
Why Is FP-Growth the Winner?

- 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

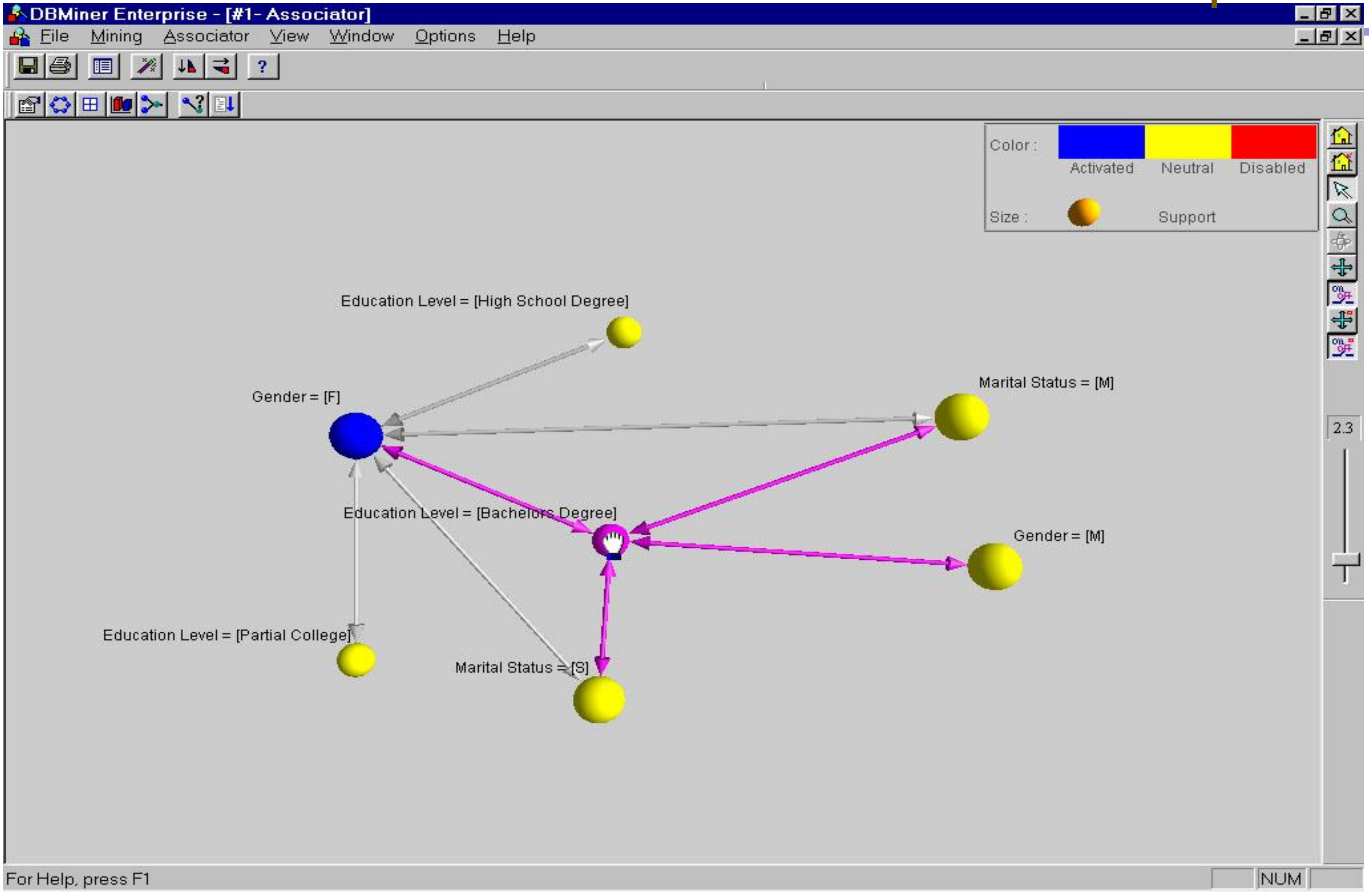
Discussion

- Advantages of FP-Growth
  - only 2 passes over data-set
  - “compresses” data-set
  - no candidate generation
  - much faster than Apriori
- Disadvantages of FP-Growth
  - FP-Tree may not fit in memory!!
  - FP-Tree is expensive to build

Visualization of Association Rules: Plane Graph



Visualization of Association Rules: Rule Graph



Visualization of Association Rules (SGI/MineSet 3.0)