Association Rule Discovery:

Mining Frequent Patterns without Candidate Generation

The FP-Growth Approach

Construction of FP-tree

- Step 1: scan the transaction database D once. Collect the set of frequent items F and their supports. Sort F in support descending order as L, the list of frequent items.
- Step 2: create the root of an FP-tree, and label it as "null". For each transaction T in D do the following:
  - select and sort the frequent items in T according to the order of L. Let the sorted frequent item list in T be [p|P], where p is the first element and P is the remaining list.

Construction of FP-tree (cont.)

Call **insert_tree**([p|P], T), which is performed as follows:

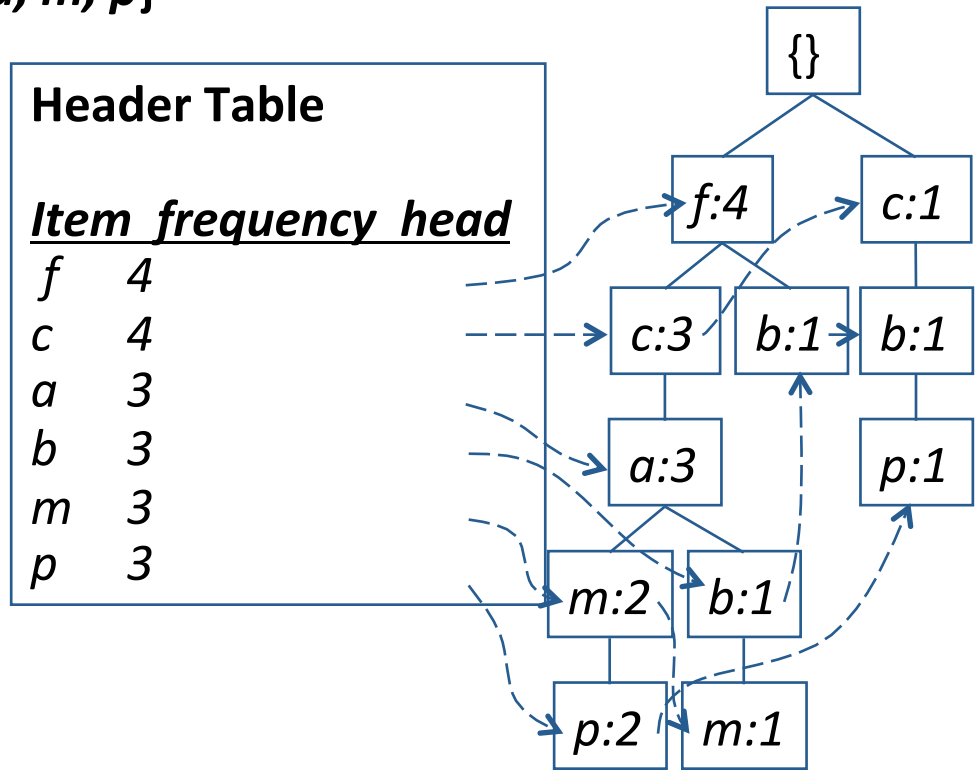
- if T has a child N such that *N.item-name = p.item-name*, then increment N's count by 1; else create a new node N, and let its count be 1, its parent link be linked to T, and its node-link to the nodes with the same item-name via the node-link structure. If P is non-empty, call **insert-tree**(P, N) recursively.

Construction of FP-tree from a Transaction Database (An example)

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_count=3

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



Benefits of the FP-tree Structure

- Completeness
  - Never breaks a long pattern of any transaction
  - Preserves complete information for frequent pattern mining
- Compactness
  - Reducing 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)
  - The compression ratio could be 20 ~ 100

Mining Frequent Patterns with FP-trees

- Idea: Recursively grow frequent patterns by pattern and database partition
- Terminology:
  - *prefix path (P) of an item a_i*: a path from the root of the FP-tree to the parent node of a_i
  - *transformed prefix path of a_i for path P :* a_i's prefix path with the frequency count of every node in P adjusted to the same as the count for a_i

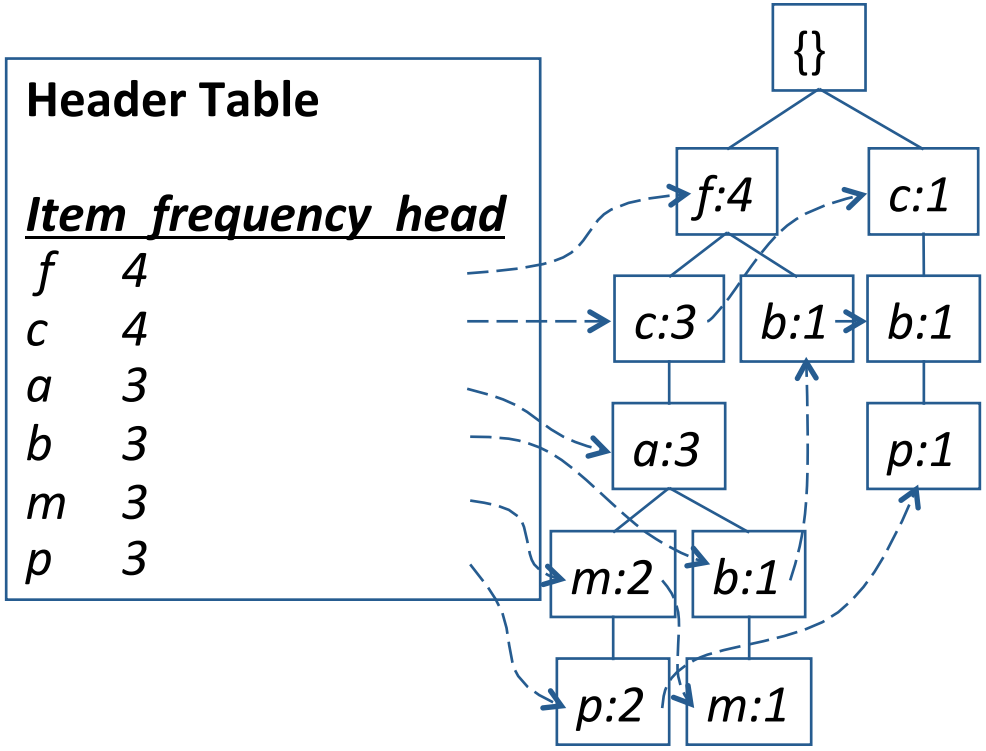
Mining Frequent Patterns with FP-trees

- – *a_i's conditional pattern base :* the set of transformed prefix paths of a_i
- – a_i' s conditional FP-tree : the FP-tree built based on a_i' s conditional pattern base

- Frequent pattern growth 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

From FP-tree to Conditional Pattern-Base

- 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

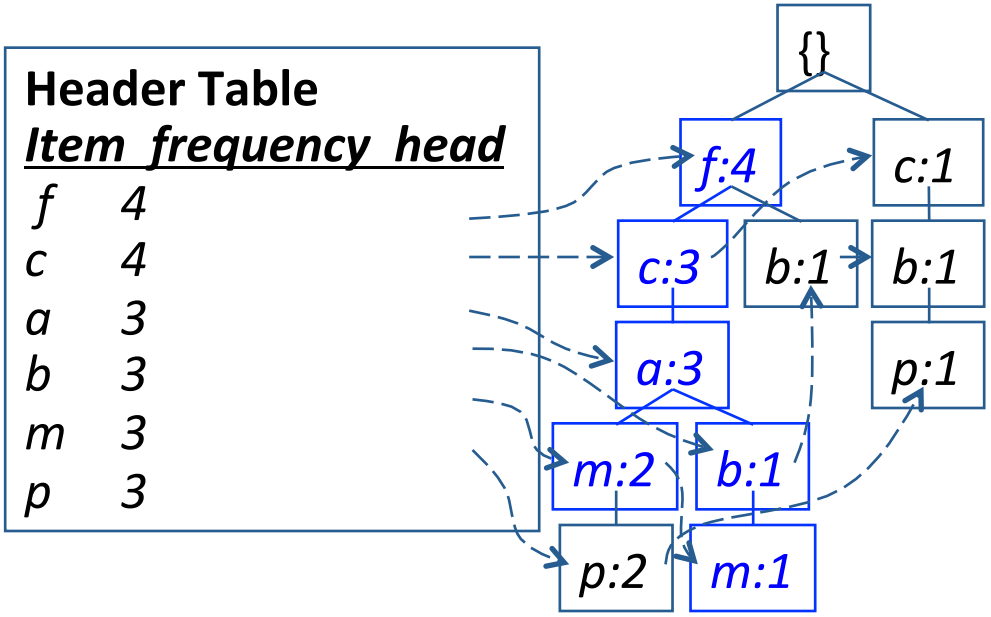


***The conditional* pattern base for each item p**

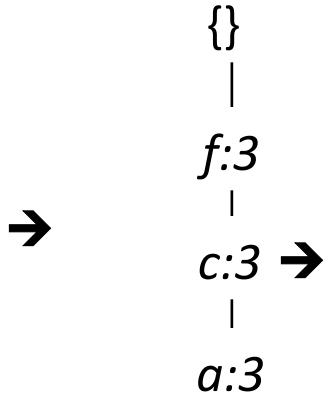
item	***cond. pattern base***
f	***4***
c	***f:3***
a	***fc:3***
b	***fca:1, f:1, c:1***
m	***fca:2, fcab:1***
p	***fcam:2, cb:1***

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's conditional* pattern base:**
fca:2, fcab:1



***M's conditional* FP-tree**

All frequent patterns relate to *m*

m,

fm, cm, am,

fcm, fam, cam,

fcam

Principles of Frequent Pattern Growth

- Pattern growth theorem
  - Let α be a frequent itemset in DB, B be α's conditional pattern base, and β be an itemset in B. Then $\alpha \cup \beta$ is a frequent itemset in DB iff β is frequent in B.
- Ex. "*abcdef*" is a frequent pattern, if and only if
  - "*abcd*" is a frequent pattern, and
  - "*ef*" is frequent in *abcd*'s conditional pattern-base (i.e., the set of transactions containing "*abcd*")

Procedure FP-growth(Tree, α) {

if Tree contains a single path P {

for each combination (β) of the nodes in P

generate patterns β union α w/ support = min_support of nodes in β

}

else for each a_i in the header of Tree {

generate pattern $\beta = a_i$ union α with support=a_i.support

construct β's conditional pattern base and then β's conditional FP_tree Tree $_\beta$

if Tree $_\beta$!= 0 then

call **FP_growth**(Tree $_\beta$, β)

}

}

Practice example

Apply FP-Growth algorithm to find all frequent itemsets from the following transaction database (minimum support = 2)

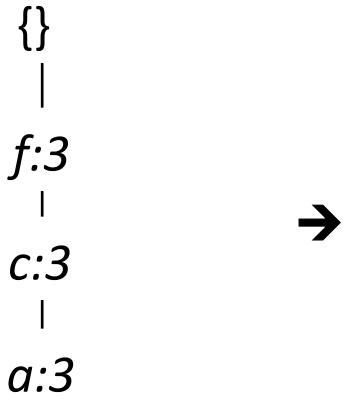
TID	list of items
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

Correctness and Completeness of Conditional Pattern-Bases

- Completeness (node-link property)
  - For any frequent item a_i, all the possible frequent patterns that contain a_i can be obtained by following a_i's node-links, starting from a_i's head in the FP-tree header
- Correctness (transformed prefix path property)
  - To calculate the frequent patterns for a node a_i in a path P, only the prefix sub-path of a_i in P need to be accumulated, and its frequency count should carry the same count as node a_i.

A Special Case: Single FP-tree Path

- Suppose a (conditional) FP-tree T has a single path P
- The complete set of frequent patterns of T can be generated by enumeration of all the combinations of the sub-paths of P



→

All frequent patterns concerning *m*

m,

fm, cm, am,

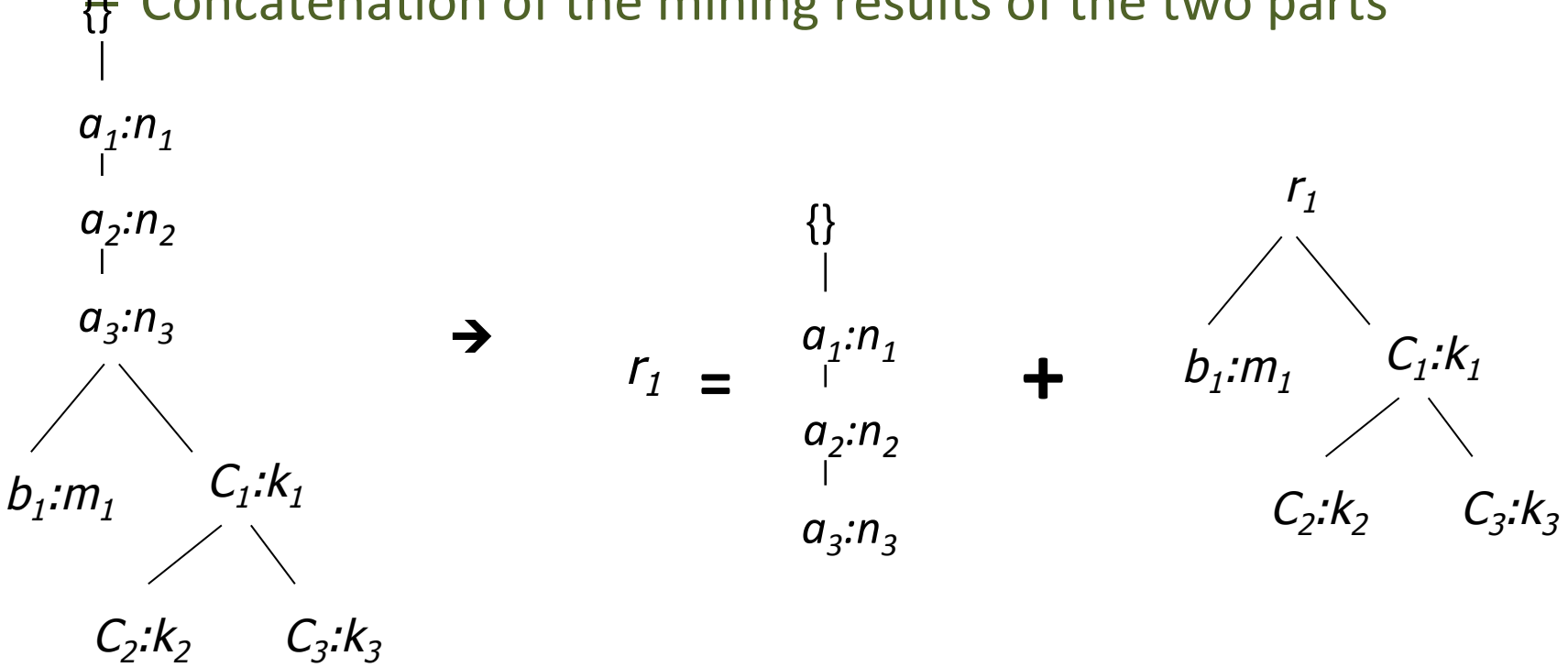
fcm, fam, cam,

fcam

***m's conditional* FP-tree**

A More General (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



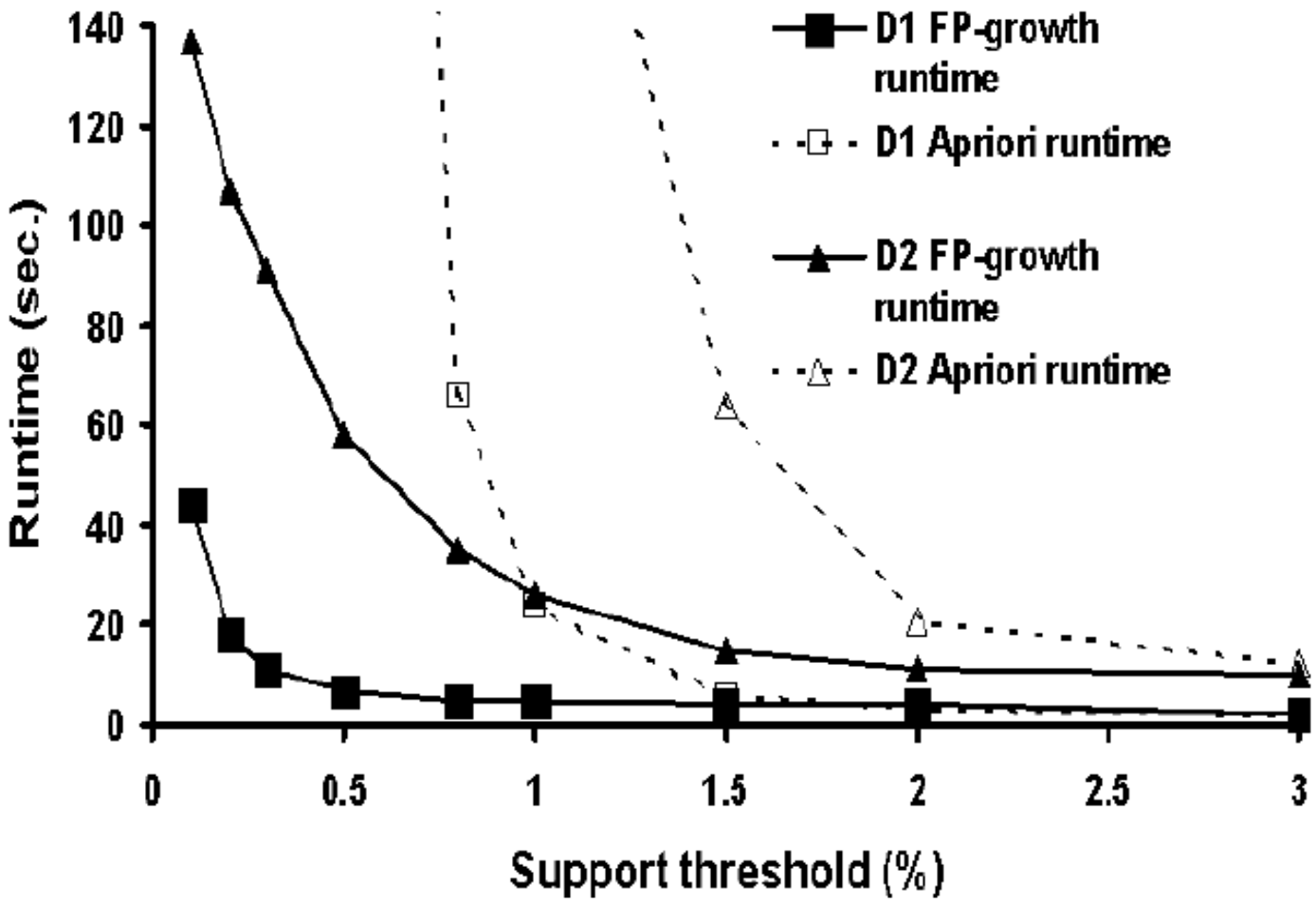
Why Is FP-Growth the Winner?

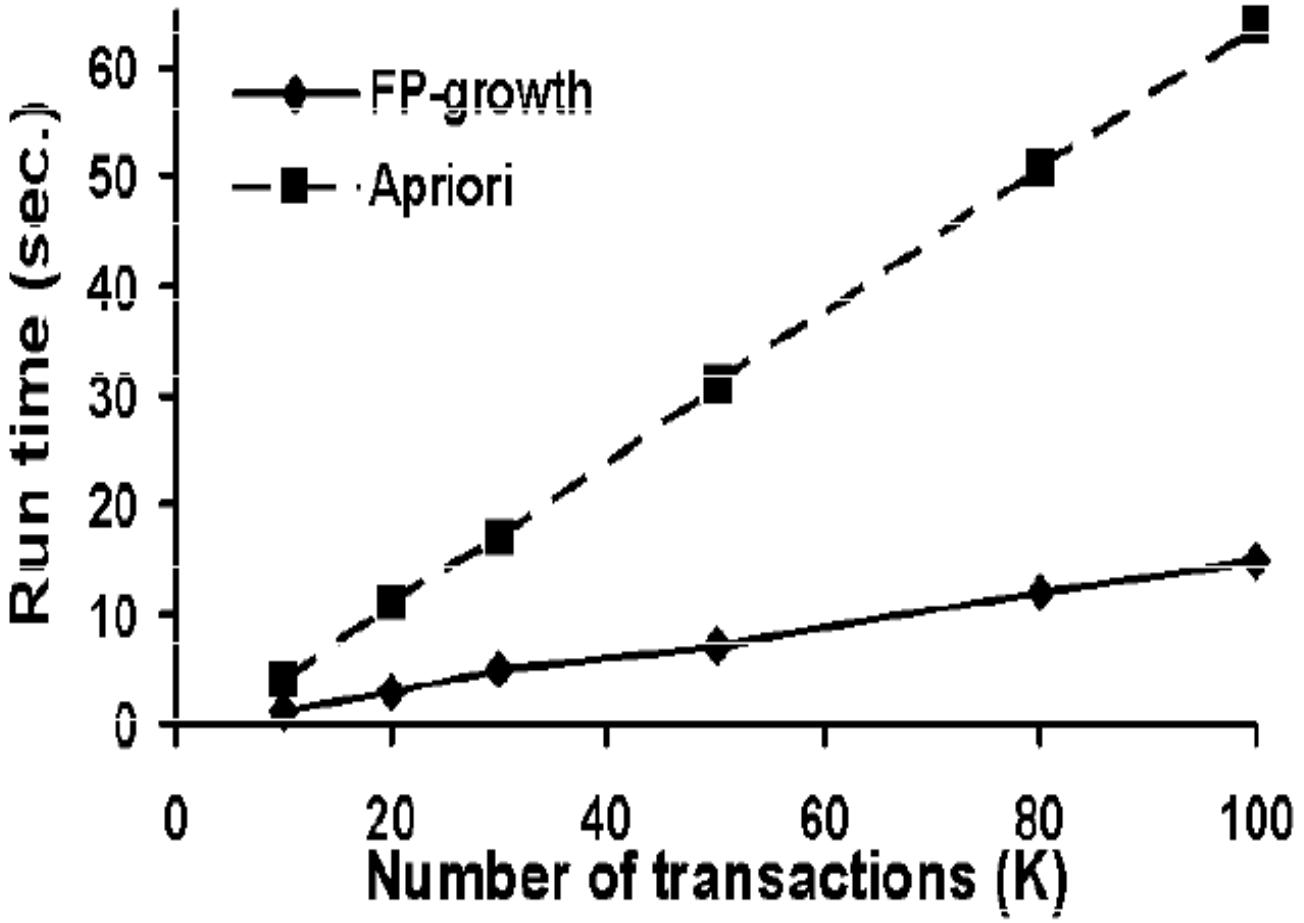
- Divide-and-conquer:
  - decompose both the mining task and DB according to the frequent patterns obtained so far
  - leads to focused search of smaller databases
- Other factors
  - no candidate generation, no candidate test
  - compressed database: FP-tree structure
  - no repeated scan of entire database
  - basic ops—counting and FP-tree building, not pattern search and matching

Experiment Setup

- Compare the runtime of FP-growth with classical Apriori
  - Runtime vs. min_sup
  - Runtime vs. size of the DB (# of transactions)

- Synthetic data sets : frequent itemsets grows exponentially as minisup goes down
  - D1: T25.I10.D10K
    - 1K items
    - avg(transaction size)=25
    - avg(max/potential frequent item size)=10
    - 10K transactions
  - D2: T25.I20.D100K
    - 10k items


140
120
100
80
60
40
20
0
Runtime (sec.)
0
0.5
1
1.5
2
2.5
3
Support threshold (%)
D1 FP-growth runtime
D1 Apriori runtime
D2 FP-growth runtime
D2 Apriori runtime


FP-growth
Apriori
Run time (sec.)
0
10
20
30
40
50
60
0
20
40
60
80
100
Number of transactions (K)