Mining Frequent Patterns Without Candidate Generation

Jiawei Han, Jian Pei and Yiwen Yin
School of Computing Science
Simon Fraser University

(Part of the slides are due to Jiawei Han)

Is Apriori Efficient Enough? □Performance Bottlenecks!!

- Basic Idea: Candidate generation-and-test
  - Use frequent (*k* □ 1)-itemsets to generate candidate frequent *k*-itemsets
  - Use database scan and pattern matching to test (i.e., collect counts for the candidate itemsets)
- Bottleneck:
  - Generation may lead to huge candidate sets
    - n frequent 1-itemset will generate n(n-1)/2 candidate 2-itemsets
    - To discover a frequent pattern of length 100, e.g., $\{a_1, a_2, □, a_{100}\}$, we need to generate $2^{100} \approx 10^{30}$ candidates.
  - Test involves multiple scans of the entire database
    - Needs $(n+1)$ scans, n is the length of the longest pattern

Can We Do Better? □ Mining frequent patterns without candidate generation

- Database projection and compression
  - Project the database based on its frequent patterns
  - Compress a database into a compact, Frequent-Pattern tree (FP-tree)
    - condensed, but complete for frequent pattern mining
    - no candidate generation: test projected database only!
- ®Divide-and-conquer□
  - decompose both the task and DB according to the frequent patterns obtained so far

Construction of FP-tree from a Transaction Database

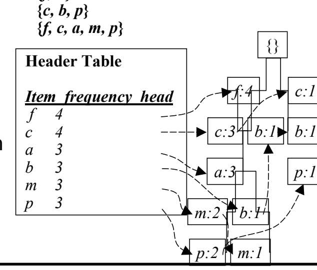
TID	*Items bought*	*(ordered) frequent items*
100	*{f, a, c, d, g, i, m, p}*	*{f, c, a, m, p}*
200	*{a, b, c, f, l, m, o}*	*{f, c, a, b, m}*
300	*{b, f, h, j, o, w}*	*{f, b}*
400	*{b, c, k, s, p}*	*{c, b, p}*
500	*{a, f, c, e, l, p, m, n}*	*{f, c, a, m, p}*

min_support = 3

1 Scan DB once, find frequent 1-itemset (single item pattern)

2 Order frequent items in frequency descending order

3 Scan DB again, construct FP-tree



Benefits of the FP-tree Structure

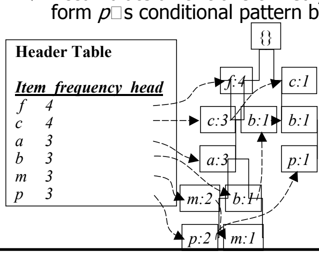
- Completeness
  - Preserves complete information for frequent pattern mining
- Compactness
  - Reducing irrelevant info□infrequent items are gone
  - Items in frequency descent order: the more frequently occurring, the more likely to be shared
  - Never be larger than the original database (not count node-links and the *count* field)
  - For Connect-4 DB, compression ratio could be over 100

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

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

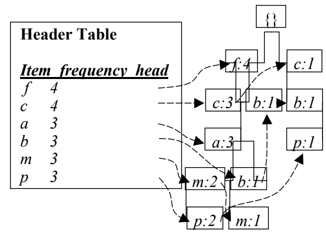


***Conditional* pattern bases**

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

Transformed Prefix Paths

- Derive the *transformed prefix paths* of item *p*
  - For each item *p* in the tree, collect *p*□s prefix path with count = *p*□s frequency

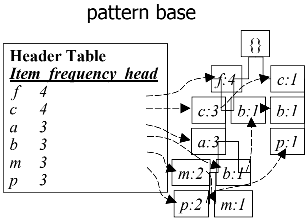


***Conditional* pattern bases**

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

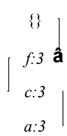
From Conditional Pattern-Bases to Conditional FP-trees

- 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

â

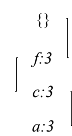


***m-conditional* FP-tree**

â

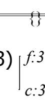
All frequent patterns relate to *m*
m,
fm, cm, am,
fcm, fam, cam,
fcam

Recursion: Mining Each Conditional FP-tree Until □



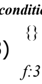
***m-conditional* FP-tree**

Cond. pattern base of ®am□: (fc:3)



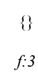
***am-conditional* FP-tree**

Cond. pattern base of ®cm□: (f:3)



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

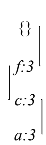
Cond. pattern base of ®cam□: (f:3)



***cam-conditional* FP-tree**

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

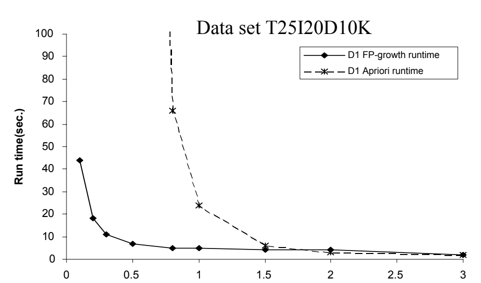


***m-conditional* FP-tree**

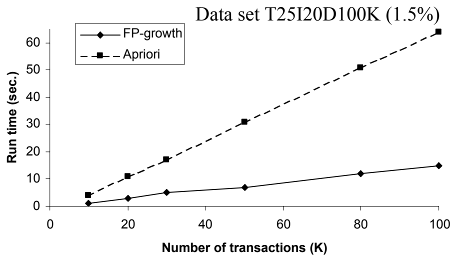
â

All frequent patterns concerning *m*
m,
fm, cm, am,
fcm, fam, cam,
fcam

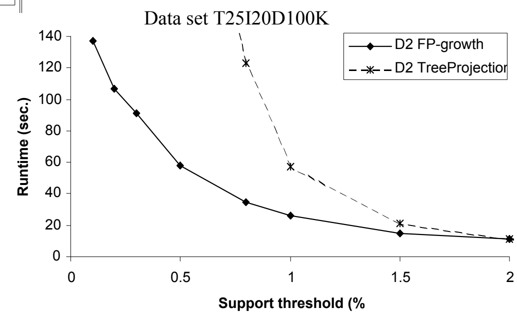
FP-Growth vs. Apriori: Scalability With the Support Threshold



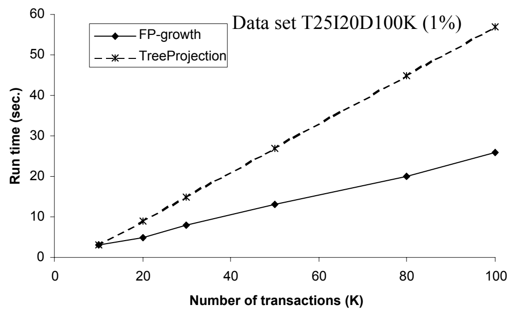
FP-Growth vs. Apriori: Scalability With the Number of Transactions



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



FP-growth vs. Tree-Projection: Scalability with the # of Transactions



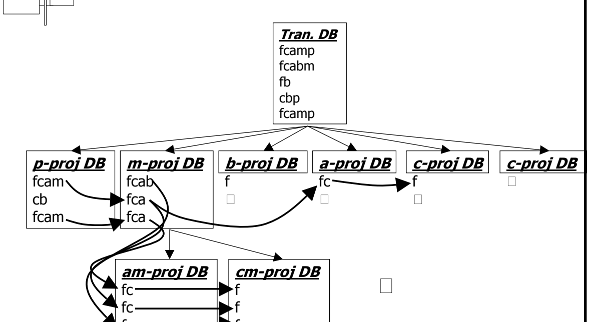
Why Is FP-Growth the Winner?

- decompose both the mining task and DB according to the frequent patterns obtained so far
- no redundant counting
- leads to focused search of smaller databases
- no candidate generation, no candidate test
- compressed database: FP-tree structure
- no repeated scan of entire database

I/O-Bound FP-Growth: 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
- Alternative methods
  - Construction of a disk-resident FP-tree
  - Materialization and incremental update of an FP-tree

Partition-Based Projection



Thank you !!!