Data Mining Techniques: Frequent Patterns in Sets and Sequences

Mirek Riedewald

Some slides based on presentations by Han/Kamber and Tan/Steinbach/Kumar

Frequent Pattern Mining Overview

- Basic Concepts and Challenges
- Efficient and Scalable Methods for Frequent Itemsets and Association Rules
- Pattern Interestingness Measures
- Sequence Mining

What Is Frequent Pattern Analysis?

- Find patterns (itemset, sequence, structure, etc.) that occur frequently in a data set
- First proposed for frequent itemsets and association rule mining
- Motivation: Find inherent regularities in data
  - What products were often purchased together?
  - What are the subsequent purchases after buying a PC?
  - What kinds of DNA are sensitive to a new drug?
- Applications
  - Market basket analysis, cross-marketing, catalog design, sale campaign analysis, Web log (click stream) analysis, DNA sequence analysis

Association Rule Mining

- Given a set of transactions, find rules that will predict the occurrence of an item based on the occurrences of other items in the transaction

Market-Basket transactions

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

Example of Association Rules

{Diaper} → {Beer},
{Milk, Bread} → {Eggs,Coke},
{Beer, Bread} → {Milk},

Implication means co-occurrence, not causality!

Definition: Frequent Itemset

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

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

Definition: Association Rule

- Association Rule = implication expression of the form X→Y, where X and Y are itemsets
  - Ex.: {Milk, Diaper} → {Beer}
- Rule Evaluation Metrics
  - Support (s) = P(X∪Y)
    - Estimated by fraction of transactions that contain both X and Y
  - Confidence (c) = P(Y| X)
    - Estimated by fraction of transactions that contain X and Y among all transactions containing X

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

Example: $\{\text{Milk}, \text{Diaper}\} \rightarrow \text{Beer}$

$$s = \frac{\sigma(\text{Milk}, \text{Diaper}, \text{Beer})}{|D|} = \frac{2}{5}$$

$$c = \frac{\sigma(\text{Milk}, \text{Diaper}, \text{Beer})}{\sigma(\text{Milk}, \text{Diaper})} = \frac{2}{3}$$

Association Rule Mining Task

- Given a transaction database DB, find all rules having support ≥ *minsup* and confidence ≥ *minconf*
- Brute-force approach:
  - List all possible association rules
  - Compute support and confidence for each rule
  - Remove rules that fail the minsup or minconf thresholds
  - Computationally prohibitive!

Mining Association Rules

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

Example rules:

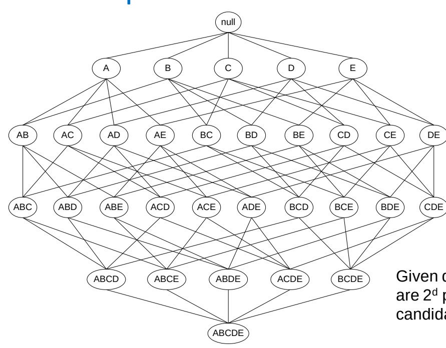
{Milk,Diaper} → {Beer} (s=0.4, c=0.67)
{Milk,Beer} → {Diaper} (s=0.4, c=1.0)
{Diaper,Beer} → {Milk} (s=0.4, c=0.67)
{Beer} → {Milk,Diaper} (s=0.4, c=0.67)
{Diaper} → {Milk,Beer} (s=0.4, c=0.5)
{Milk} → {Diaper,Beer} (s=0.4, c=0.5)

Observations:
- All the above rules are binary partitions of the same itemset {Milk, Diaper, Beer}
- Rules originating from the same itemset have identical support but can have different confidence
- Thus, we may decouple the support and confidence requirements

Mining Association Rules

- Two-step approach:
  1. Frequent Itemset Generation
     - Generate all itemsets that have support ≥ minsup
  2. Rule Generation
     - Generate high-confidence rules from each frequent itemset, where each rule is a binary partitioning of the frequent itemset
- Frequent itemset generation is still computationally expensive

Frequent Itemset Generation



Given d items, there are 2^d possible candidate itemsets

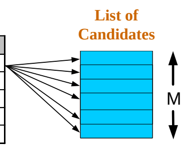
Frequent Itemset Generation

- Brute-force approach:
  - Each itemset in the lattice is a candidate frequent itemset
  - Count the support of each candidate by scanning the database
  - Match each transaction against every candidate
  - Complexity ~ O(N*M*w) => expensive since $M=2^d$

Transactions

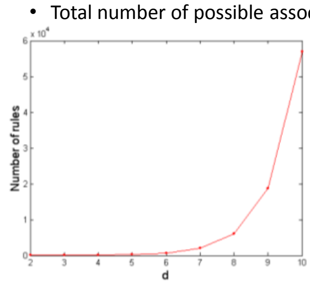
TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

N ← w →



Computational Complexity

- Given d unique items, total number of itemsets = 2^d
- Total number of possible association rules?



$$R = \sum_{k=1}^{d-1}\left[\binom{d}{k}\cdot\sum_{j=1}^{d-k}\binom{d-k}{j}\right]$$
$$= 3^d - 2^{d+1} + 1$$

If d=6, R = 602 possible rules

Frequent Pattern Mining Overview

- Basic Concepts and Challenges
- Efficient and Scalable Methods for Frequent Itemsets and Association Rules
- Pattern Interestingness Measures
- Sequence Mining

Frequent Itemset Generation Strategies

- Reduce the number of candidates (M)
  - Complete search: $M=2^d$
  - Use pruning techniques to reduce M
- Reduce the number of transactions (N)
  - Skip short transactions as size of itemset increases
- Reduce the number of comparisons (N*M)
  - Use efficient data structures to store the candidates or transactions
  - No need to match every candidate against every transaction

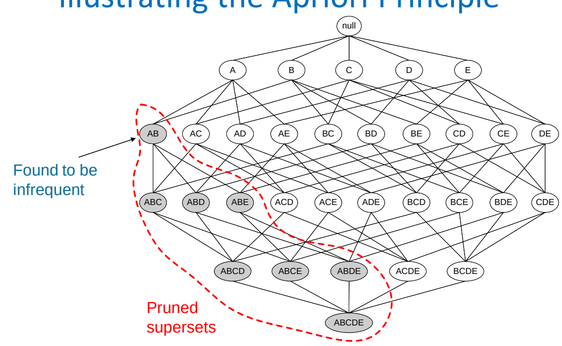
Reducing Number of Candidates

- Apriori principle:
  - If an itemset is frequent, then all of its subsets must also be frequent
- Apriori principle holds due to the following property of the support measure:

$$\forall X, Y : (X \subseteq Y) \Rightarrow s(X) \geq s(Y)$$

  - Support of an itemset never exceeds the support of its subsets
  - This is known as the anti-monotone property of support

Illustrating the Apriori Principle



Illustrating the Apriori Principle

Items (1-itemsets)

Item	Count
Bread	4
Coke	2
Milk	4
Beer	3
Diaper	4
Eggs	1

Pairs (2-itemsets)

Itemset	Count
{Bread,Milk}	3
{Bread,Beer}	2
{Bread,Diaper}	3
{Milk,Beer}	2
{Milk,Diaper}	3
{Beer,Diaper}	3

(No need to generate candidates involving Coke or Eggs)

Minimum Support = 3

Triplets (3-itemsets)

Itemset	Count
{Bread,Milk,Diaper}	3

If every subset is considered,
$^6C_1 + {}^6C_2 + {}^6C_3 = 41$
With support-based pruning,
6 + 6 + 1 = 13

Apriori Algorithm

- Generate L_1 = frequent itemsets of length k=1
- Repeat until no new frequent itemsets are found
  - Generate C_{k+1}, the length-(k+1) candidate itemsets, from L_k
  - Prune candidate itemsets in C_{k+1} containing subsets of length k that are not in L_k (and hence infrequent)
  - Count support of each remaining candidate by scanning DB; eliminate infrequent ones from C_{k+1}
  - $L_{k+1}=C_{k+1}$; k = k+1

Important Details of Apriori

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

- Example of Candidate-generation for L_3={ {a,b,c}, {a,b,d}, {a,c,d}, {a,c,e}, {b,c,d} }
  - Self-joining L_3
    - {a,b,c,d} from {a,b,c} and {a,b,d}
    - {a,c,d,e} from {a,c,d} and {a,c,e}
  - Pruning:
    - {a,c,d,e} is removed because {a,d,e} is not in L_3
  - C_4={ {a,b,c,d} }

19

How to Generate Candidates?

- Step 1: self-joining L_{k-1}

  insert into C_k
  select p.item$_1$, p.item$_2$,…, p.item$_{k-1}$, q.item$_{k-1}$
  from L_{k-1} p, L_{k-1} q
  where p.item$_1$=q.item$_1$ AND … AND p.item$_{k-2}$=q.item$_{k-2}$ AND p.item$_{k-1}$ < q.item$_{k-1}$

- Step 2: pruning
  - forall itemsets c in 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

20

How to Count Supports of Candidates?

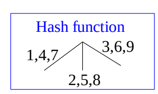
- Why is counting supports of candidates a problem?
  - Total number of candidates can be very large
  - One transaction may contain many candidates

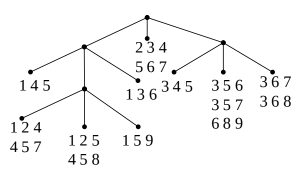
- Method:
  - Candidate itemsets stored in a hash-tree
  - Leaf node contains list of itemsets
  - Interior node contains a hash table
  - Subset function finds all candidates contained in a transaction

21

Generate Hash Tree

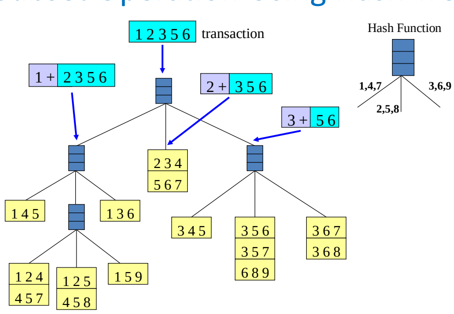
- Suppose we have 15 candidate itemsets of length 3:
  - {1 4 5}, {1 2 4}, {4 5 7}, {1 2 5}, {4 5 8}, {1 5 9}, {1 3 6}, {2 3 4}, {5 6 7}, {3 4 5}, {3 5 6}, {3 5 7}, {6 8 9}, {3 6 7}, {3 6 8}
- We need:
  - Hash function
  - Max leaf size: max number of itemsets stored in a leaf node (if number of candidate itemsets exceeds max leaf size, split the node)





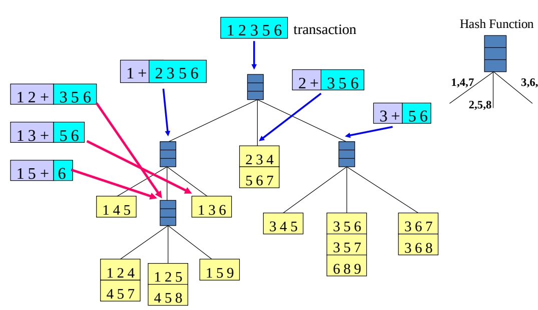
22

Subset Operation Using Hash Tree



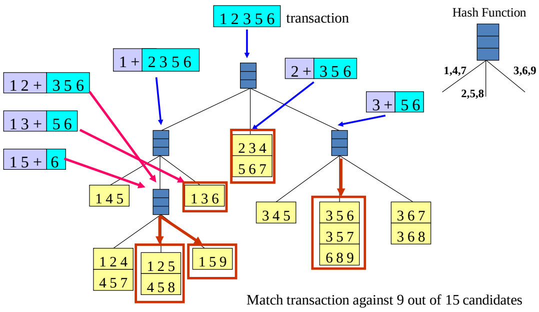
23

Subset Operation Using Hash Tree



24

Subset Operation Using Hash Tree



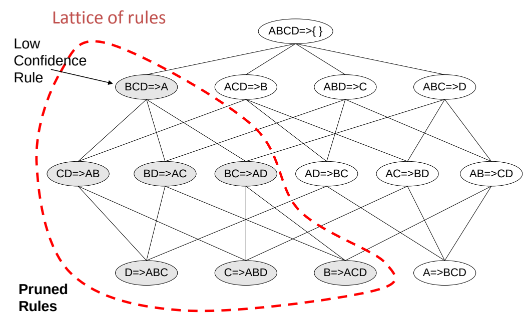
Association Rule Generation

- Given a frequent itemset L, find all non-empty subsets $f \subset L$ such that $f \rightarrow L - f$ satisfies the minimum confidence requirement
  - If {A,B,C,D} is a frequent itemset, candidate rules are:
    - ABC →D, ABD →C, ACD →B, BCD →A, A →BCD, B →ACD, C →ABD, D →ABC, AB →CD, AC → BD, AD → BC, BC →AD, BD →AC, CD →AB
- If |L| = k, then there are $2^k - 2$ candidate association rules (ignoring $L \rightarrow \emptyset$ and $\emptyset \rightarrow L$)

Rule Generation

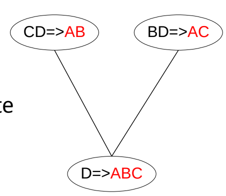
- How do we efficiently generate association rules from frequent itemsets?
  - In general, confidence does not have an anti-monotone property
    - c(ABC→D) can be larger or smaller than c(AB→D)
  - But confidence of rules generated from the same itemset has an anti-monotone property
    - For {A,B,C,D}, $c(ABC \rightarrow D) \geq c(AB \rightarrow CD) \geq c(A \rightarrow BCD)$
    - Confidence is anti-monotone w.r.t. number of items on the right-hand side of the rule

Rule Generation for Apriori Algorithm



Rule Generation for Apriori Algorithm

- Candidate rule is generated by merging two rules that share the same prefix in the rule consequent
- Join(CD→AB, BD→AC) would produce the candidate rule D → ABC
- Prune rule D→ABC if its subset AD→BC does not have high confidence



Improving Apriori

- Challenges
  - Multiple scans of transaction database
  - Huge number of candidates
  - Tedious workload of support counting for candidates
- General ideas
  - Reduce passes of transaction database scans
  - Further shrink number of candidates
  - Facilitate support counting of candidates

Bottleneck of Frequent-Pattern Mining

- Apriori generates a very large number of candidates
  - 10^4 frequent 1-itemsets can result in more than 10^7 candidate 2-itemsets
  - Many candidates might have low support, or do not even exist in the database
- Apriori scans entire transaction database for every round of support counting
- Bottleneck: candidate-generation-and-test
- Can we avoid candidate generation?

How to Avoid Candidate Generation

- Grow long patterns from short ones using local frequent items
  - Assume {a,b,c} is a frequent pattern in transaction database DB
  - Get all transactions containing {a,b,c}
    - Notation: DB|{a,b,c}
  - {d} is a local frequent item in DB|{a,b,c}, if and only if {a,b,c,d} is a frequent pattern in DB

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-itemsets (single item pattern)
2. Sort frequent items in frequency descending order, get 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

Construct FP-tree from a Transaction Database

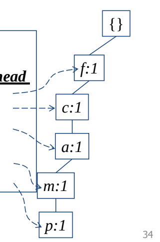
TID	Items bought	(ordered) frequent items
100	{f, a, c, d, g, i, m, p}	{f, c, a, m, p}
200	{a, b, c, f, l, m, o}	{f, c, a, b, m}
300	{b, f, h, j, o, w}	{f, b}
400	{b, c, k, s, p}	{c, b, p}
500	{a, f, c, e, l, p, m, n}	{f, c, a, m, p}

min_support = 3

1. Scan DB once, find frequent 1-itemsets (single item pattern)
2. Sort frequent items in frequency descending order, get 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

Construct FP-tree from a Transaction Database

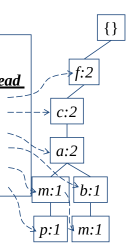
TID	Items bought	(ordered) frequent items
100	{f, a, c, d, g, i, m, p}	{f, c, a, m, p}
200	{a, b, c, f, l, m, o}	{f, c, a, b, m}
300	{b, f, h, j, o, w}	{f, b}
400	{b, c, k, s, p}	{c, b, p}
500	{a, f, c, e, l, p, m, n}	{f, c, a, m, p}

min_support = 3

1. Scan DB once, find frequent 1-itemsets (single item pattern)
2. Sort frequent items in frequency descending order, get 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

Construct FP-tree from a Transaction Database

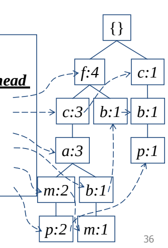
TID	Items bought	(ordered) frequent items
100	{f, a, c, d, g, i, m, p}	{f, c, a, m, p}
200	{a, b, c, f, l, m, o}	{f, c, a, b, m}
300	{b, f, h, j, o, w}	{f, b}
400	{b, c, k, s, p}	{c, b, p}
500	{a, f, c, e, l, p, m, n}	{f, c, a, m, p}

min_support = 3

1. Scan DB once, find frequent 1-itemsets (single item pattern)
2. Sort frequent items in frequency descending order, get 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 larger than the original database (if we do not count node-links and the count field)
  - For some example DBs, compression ratio over 100

Partition Patterns and Databases

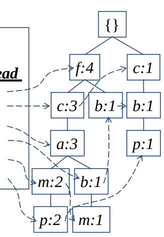
- Frequent patterns can be partitioned into subsets according to f-list
  - F-list=f-c-a-b-m-p
  - Patterns containing p
  - Patterns having m, but no p
  - Patterns having b, but neither m nor p
  - …
  - Patterns having c, but neither a, b, m, nor p
  - Pattern f
- This partitioning is complete and non-redundant

Construct Conditional Pattern Base For Item X

- Conditional pattern base = set of prefix paths in FP-tree that co-occur with x
- Traverse FP-tree by following link of frequent item x in header table
- Accumulate paths with their frequency counts

Header Table

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



***Conditional* pattern bases**

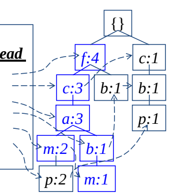
item	*cond. pattern base*
c	*f:3*
a	*fc:3*
b	*fca:1, f:1, c:1*
m	*fca:2, fcab:1*
p	*fcam:2, cb:1*

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

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

All frequent patterns having *m*, but not *p*
m,
fm, cm, am,
fcm, fam, cam,
fcam

Recursion: Mining Conditional FP-Trees

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

Output: am
Cond. pattern base of "am": *fc:3*

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

Output: cm
Cond. pattern base of "cm": *f:3*

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

Output: fm
Cond. pattern base of "fm": *{}*

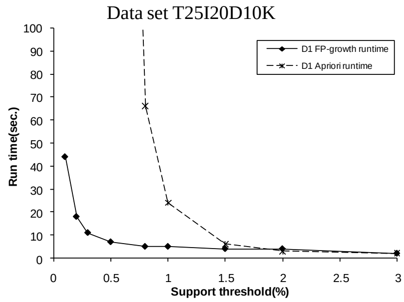
For am-conditional FP-tree, output cam
Cond. pattern base of "cam": *f:3*

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

FP-Tree Algorithm Summary

- 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 recursively on each newly created conditional FP-tree
  - Stop recursion when resulting FP-tree is empty
    - Optimization if tree contains only one path: single path will generate all the combinations of its sub-paths, each of which is a frequent pattern

FP-Growth vs. Apriori: Scalability With 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 operations: counting local frequent single items and building sub FP-tree
    - No pattern search and matching

Factors Affecting Mining Cost

- Choice of minimum support threshold
  - Lower support threshold => more frequent itemsets
    - More candidates, longer frequent itemsets
- Dimensionality (number of items) of the data set
  - More space needed to store support count of each item
  - If number of frequent items also increases, both computation and I/O costs may increase
- Size of database
  - Each pass over DB is more expensive
- Average transaction width
  - May increase max. length of frequent itemsets and traversals of hash tree (more subsets supported by transaction)

- How can we further reduce some of these costs?

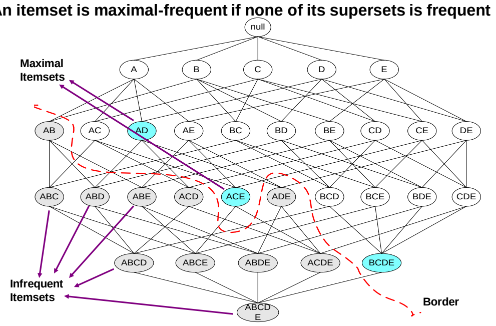
Compact Representation of Frequent Itemsets

- Some itemsets are redundant because they have identical support as their supersets
- Number of frequent itemsets $= 3 \times \sum_{k=1}^{10} \binom{10}{k}$
- Need a compact representation

TID	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1

Maximal Frequent Itemset

An itemset is maximal-frequent if none of its supersets is frequent



Closed Itemset

- A frequent itemset is closed if none of its supersets has the same support
  - Lossless compression of the set of all frequent itemsets

TID	Items
1	{A,B}
2	{B,C,D}
3	{A,B,C,D}
4	{A,B,D}
5	{A,B,C,D}

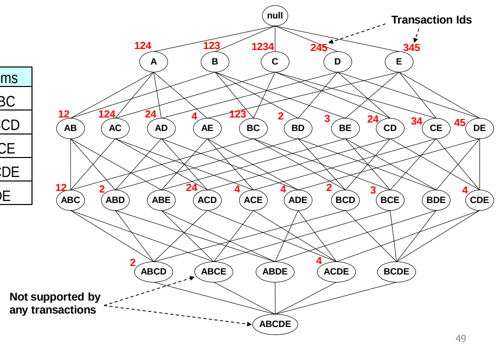
min_sup = 2

Itemset	Support
{A}	4
{B}	5
{C}	3
{D}	4
{A,B}	4
{A,C}	2
{A,D}	3
{B,C}	3
{B,D}	4
{C,D}	3

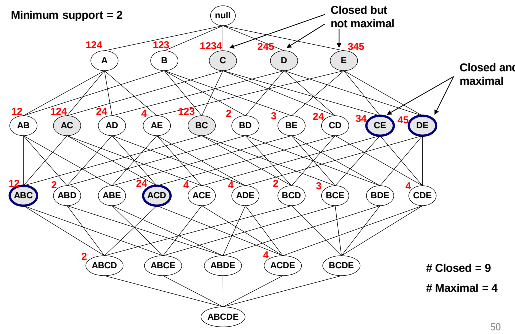
Itemset	Support
{A,B,C}	2
{A,B,D}	3
{A,C,D}	2
{B,C,D}	3
{A,B,C,D}	2

Maximal vs Closed Frequent Itemsets

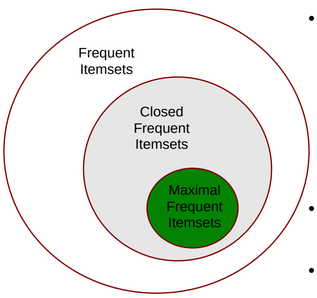
TID	Items
1	ABC
2	ABCD
3	BCE
4	ACDE
5	DE



Maximal vs Closed Frequent Itemsets



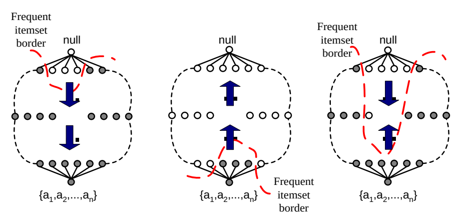
Maximal vs Closed Frequent Itemsets



- How to efficiently find maximal frequent itemsets? (similar for closed ones)
  - Naïve: first find all frequent itemsets, then remove non-maximal ones
  - Better: use maximality property for pruning
- Effectiveness depends on itemset generation strategy
- See book for details

Methods for Frequent Itemset Generation

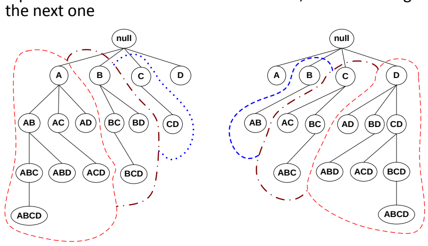
- Traversal of itemset lattice
  - General-to-specific: Apriori
  - Specific-to-general: good for pruning for maximal frequent itemsets



(a) General-to-specific (b) Specific-to-general (c) Bidirectional

Alternative Methods for Frequent Itemset Generation

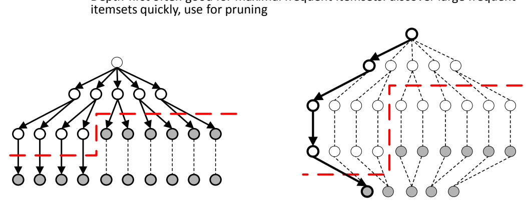
- Traversal of itemset lattice
  - Equivalence Classes: search one class first, before moving on to the next one



(a) Prefix tree (b) Suffix tree

Alternative Methods for Frequent Itemset Generation

- Traversal of Itemset Lattice
  - Breadth-first vs Depth-first
    - Apriori is breadth-first (good for pruning)
    - Depth-first often good for maximal frequent itemsets: discover large frequent itemsets quickly, use for pruning

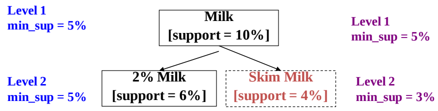


(a) Breadth first (b) Depth first

Extension: Mining Multiple-Level Association Rules

- Items often form hierarchies
  - Most relevant pattern might only show at the right granularity
- Flexible support settings
  - Items at the lower level are expected to have lower support

uniform support | reduced support



Extension: Mining Multi-Dimensional Associations

- Single-dimensional rules: one type of predicate
  - buys(X, "milk") → buys(X, "bread")
- Multi-dimensional rules: ≥ 2 types of predicates
  - Interdimensional association rules (no repeated predicates)
    - age(X, "19-25") ∧ occupation(X, "student") → buys(X, "coke")
  - Hybrid-dimensional association rules (repeated predicates)
    - age(X, "19-25") ∧ buys(X, "popcorn") → buys(X, "coke")
- See book for efficient mining algorithms

Frequent Pattern Mining Overview

- Basic Concepts and Challenges
- Efficient and Scalable Methods for Frequent Itemsets and Association Rules
- Pattern Interestingness Measures
- Sequence Mining

Lift

- Rule found: Basketball →Cereal [40%, 66.7%]
  - Misleading, because overall % of students eating cereal is 75% (which is > 66.7%)
- Basketball→Not_cereal [20%, 33.3%] is more useful, although with lower support and confidence
- Measure of dependent/correlated events: lift

$$\text{lift}(A,B) = \frac{P(A \cup B)}{P(A)P(B)}$$

A, B are itemsets

	Basketball	Not basketball	Sum (row)
Cereal	2000	1750	3750
Not cereal	1000	250	1250
Sum(col.)	3000	2000	5000

$$lift(B,C) = \frac{2000/5000}{3000/5000 * 3750/5000} = 0.89 \qquad lift(B,\neg C) = \frac{1000/5000}{3000/5000 * 1250/5000} = 1.33$$

Lift vs. Other Correlation Measures

- Intuition: Are milk and coffee usually bought together?
  - (m, c) > (~m, c) + (m, ~c)
- m and c are…
  - bought together in A's
  - independent in B
  - not bought together in C's
- All measures good for B
- Lift, χ^2 bad for A's, C's
  - Reason: strongly affected by number of null-transactions (those without m, c)
- all_conf, cosine good for A's, C's
  - Not affected by number of null-transactions

	Milk	No Milk
Coffee	m, c	~m, c
No Coffee	m, ~c	~m, ~c

$$\text{all\_conf}(A) = \frac{\sup(A)}{\text{max\_item\_sup}(A)}$$

$$\text{cosine}(A,B) = \frac{P(A \cup B)}{\sqrt{P(A)P(B)}}$$

Lift vs. cosine: cosine does not depend on size of DB

Data Set	*mc*	$\overline{m}c$	$m\overline{c}$	$\overline{mc}$	*all_conf.*	*cosine*	*lift*	X^2
A_1	1,000	100	100	100,000	0.91	0.91	83.64	83,452.6
A_2	1,000	100	100	10,000	0.91	0.91	9.26	9,055.7
A_3	1,000	100	100	1,000	0.91	0.91	1.82	1,472.7
A_4	1,000	100	100	0	0.91	0.91	0.99	9.9
B_1	1,000	1,000	1,000	1,000	0.50	0.50	1.00	0.0
C_1	100	1,000	1,000	100,000	0.09	0.09	8.44	670.0
C_2	1,000	100	10,000	100,000	0.09	0.29	9.18	8,172.8
C_3	1	1	100	10,000	0.01	0.07	50.0	48.5

Which Measure Is Best?

- Does it identify the right patterns?
- Does it result in an efficient mining algorithm?

symbol	measure	range	formula
ϕ	ϕ-coefficient	-1…1	$\frac{P(A,B)-P(A)P(B)}{\sqrt{P(A)P(B)(1-P(A))(1-P(B))}}$
Q	Yule's Q	-1 … 1	$\frac{P(A,B)P(\overline{A},\overline{B})-P(A,\overline{B})P(\overline{A},B)}{P(A,B)P(\overline{A},\overline{B})+P(A,\overline{B})P(\overline{A},B)}$
Y	Yule's Y	-1 … 1	$\frac{\sqrt{P(A,B)P(\overline{A},\overline{B})}-\sqrt{P(A,\overline{B})P(\overline{A},B)}}{\sqrt{P(A,B)P(\overline{A},\overline{B})}+\sqrt{P(A,\overline{B})P(\overline{A},B)}}$
k	Cohen's	-1 … 1	$\frac{P(A,B)+P(\overline{A},\overline{B})-P(A)P(B)-P(\overline{A})P(\overline{B})}{1-P(A)P(B)-P(\overline{A})P(\overline{B})}$
PS	Piatetsky-Shapiro's	-0.25 … 0.25	$P(A,B)-P(A)P(B)$
F	Certainty factor	-1 … 1	$\max(\frac{P(B\|A)-P(B)}{1-P(B)}, \frac{P(A\|B)-P(A)}{1-P(A)})$
AV	added value	-0.5 … 1	$\max(P(B\|A)-P(B), P(A\|B)-P(A))$
K	Klosgen's Q	-0.33 … 0.38	$\sqrt{P(A,B)}\max(P(B\|A)-P(B), P(A\|B)-P(A))$
g	Goodman-kruskal's	0 … 1	$\frac{\Sigma_j \max_k P(A_j,B_k)+\Sigma_k \max_j P(A_j,B_k)-\max_j P(A_j)-\max_k P(B_k)}{2-\max_j P(A_j)-\max_k P(B_k)}$
M	Mutual Information	0 … 1	$\frac{\Sigma_i\Sigma_j P(A_i,B_j)\log\frac{P(A_i,B_j)}{P(A_i)P(B_j)}}{\min(-\Sigma_i P(A_i)\log P(A_i)\log P(A_i), -\Sigma_i P(B_i)\log P(B_i)\log P(B_i))}$
J	J-Measure	0 … 1	$\max(P(A,B)\log(\frac{P(B\|A)}{P(B)})+P(A\overline{B})\log(\frac{P(\overline{B}\|A)}{P(\overline{B})}),\ P(A,B)\log(\frac{P(A\|B)}{P(A)})+P(\overline{A}B)\log(\frac{P(\overline{A}\|B)}{P(\overline{A})}))$
G	Gini index	0 … 1	$\max(P(A)[P(B\|A)^2+P(\overline{B}\|A)^2]+P(\overline{A})[P(B\|\overline{A})^2+P(\overline{B}\|\overline{A})^2]-P(B)^2-P(\overline{B})^2,\ P(B)[P(A\|B)^2+P(\overline{A}\|B)^2]+P(\overline{B})[P(A\|\overline{B})^2+P(\overline{A}\|\overline{B})^2]-P(A)^2-P(\overline{A})^2)$
s	support	0 … 1	$P(A,B)$
c	confidence	0 … 1	$max(P(B\|A), P(A\|B))$
L	Laplace	0 … 1	$\max(\frac{NP(A,B)+1}{NP(A)+2}, \frac{NP(A,B)+1}{NP(B)+2})$
IS	Cosine	0 … 1	$\frac{P(A,B)}{\sqrt{P(A)P(B)}}$
γ	coherence(Jaccard)	0 … 1	$\frac{P(A,B)}{P(A)+P(B)-P(A,B)}$
α	all_confidence	0 … 1	$\frac{P(A,B)}{\max(P(A),P(B))}$
o	odds ratio	0 … ∞	$\frac{P(A,B)P(\overline{A},\overline{B})}{P(\overline{A},B)P(A,\overline{B})}$
V	Conviction	0.5 … ∞	$\max(\frac{P(A)P(\overline{B})}{P(A\overline{B})}, \frac{P(B)P(\overline{A})}{P(B\overline{A})})$
λ	lift	0 … ∞	$\frac{P(A,B)}{P(A)P(B)}$
S	Collective strength	0 … ∞	$\frac{P(A,B)+P(\overline{AB})}{P(A)P(B)+P(\overline{A})P(\overline{B})} \times \frac{1-P(A)P(B)-P(\overline{A})P(\overline{B})}{1-P(A,B)-P(\overline{AB})}$
χ^2	χ^2	0 … ∞	$\Sigma_i \frac{(P(A_i)-E_i)^2}{E_i}$

Symbol	Measure	Range	P1	P2	P3	O1	O2	O3	O3'	O4
Φ	Correlation	-1 … 0 … 1	Yes	Yes	Yes	Yes	No	Yes	Yes	No
λ	Lambda	0 … 1	Yes	No	No	Yes	No	No*	Yes	No
α	Odds ratio	0 … 1 … ∞	Yes*	Yes	Yes	Yes	Yes	Yes*	Yes	No
Q	Yule's Q	-1 … 0 … 1	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
Y	Yule's Y	-1 … 0 … 1	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
κ	Cohen's	-1 … 0 … 1	Yes	Yes	Yes	Yes	No	No	Yes	No
M	Mutual Information	0 … 1	Yes	Yes	Yes	Yes	No	No*	Yes	No
J	J-Measure	0 … 1	Yes	No	No	No	No	No	No	No
G	Gini Index	0 … 1	Yes	No	No	No	No	No*	Yes	No
s	Support	0 … 1	No	Yes	No	Yes	No	No	No	No
c	Confidence	0 … 1	No	Yes	No	Yes	No	No	No	Yes
L	Laplace	0 … 1	No	Yes	No	Yes	No	No	No	No
V	Conviction	0.5 … 1 … ∞	No	Yes	No	Yes**	No	No	Yes	No
I	Interest	0 … 1 … ∞	Yes*	Yes	Yes	Yes	No	No	No	No
IS	IS (cosine)	0 .. 1	No	Yes	Yes	Yes	No	No	No	Yes
PS	Piatetsky-Shapiro's	-0.25 … 0 … 0.25	Yes	Yes	Yes	Yes	No	Yes	Yes	No
F	Certainty factor	-1 … 0 … 1	Yes	Yes	Yes	No	No	No	Yes	No
AV	Added value	0.5 … 1 … 1	Yes	Yes	Yes	No	No	No	No	No
S	Collective strength	0 … 1 … ∞	No	Yes	Yes	Yes	No	Yes*	Yes	No
ζ	Jaccard	0 .. 1	No	Yes	Yes	Yes	No	No	No	Yes
K	Klosgen's	$\left(\sqrt{\frac{2}{\sqrt{3}}}-1\right)\left(2-\sqrt{3}-\frac{1}{\sqrt{3}}\right)\ldots 0 \ldots \frac{2}{3\sqrt{3}}$	Yes	Yes	Yes	No	No	No	No	No

The P's and O's are various desirable properties, e.g., symmetry under variable permutation (O1), which we do not cover in this class. Take-away message: no interestingness measure has all the desirable properties.

Frequent Pattern Mining Overview

- Basic Concepts and Challenges
- Efficient and Scalable Methods for Frequent Itemsets and Association Rules
- Pattern Interestingness Measures
- Sequence Mining

Introduction

- Sequence mining is relevant for transaction databases, time-series databases, and sequence databases
- Applications of sequential pattern mining
  - Customer shopping sequences:
    - First buy computer, then CD-ROM, and then digital camera, within 3 months
  - Medical treatments, natural disasters (e.g., earthquakes), science & engineering processes, stocks and markets
  - Telephone calling patterns, Weblog click streams
  - DNA sequences and gene structures

What Is Sequential Pattern Mining?

- Given a set of sequences, find all frequent subsequences

A sequence: <(ef) (ab) (df) c b >

A sequence database

SID	sequence
10	<a(abc)(ac)d(cf)>
20	<(ad)c(bc)(ae)>
30	<(ef)(ab)(df)cb>
40	<eg(af)cbc>

An element may contain a set of items. Items within an element are unordered and we list them alphabetically

<a(bc)dc> is a subsequence of <a(abc)(ac)d(cf)>

Given support threshold *min_sup* =2, <(ab)c> is a sequential pattern

Challenges of Sequential Pattern Mining

- Huge number of possible patterns
- A mining algorithm should
  - find all patterns satisfying the minimum support threshold
  - be highly efficient and scalable
  - be able to incorporate user-specific constraints

Apriori Property of Sequential Patterns

- If a sequence S is not frequent, then none of the super-sequences of S is frequent
  - E.g, if <hb> is infrequent, then so are <hab> and <(ah)b>

Seq. ID	Sequence
10	<(bd)cb(ac)>
20	<(bf)(ce)b(fg)>
30	<(ah)(bf)abf>
40	<(be)(ce)d>
50	<a(bd)bcb(ade)>

Given support threshold *min_sup* =2, find all frequent subsequences

GSP: Generalized Sequential Pattern Mining

- Initially, every item in DB is a candidate of length k=1
- For each level (i.e., sequences of length k) do
  - Scan database to collect support count for each candidate sequence
  - Generate candidate length-(k+1) sequences from length-k frequent sequences
    - Join phase: sequences s_1 and s_2 join, if s_1 without its first item is identical to s_2 without its last item
    - Prune phase: delete candidates that contain a length-k subsequence that is not among the frequent ones
- Repeat until no frequent sequence or no candidate can be found
- Major strength: Candidate pruning by Apriori

80

Finding Length-1 Sequential Patterns

- Initial candidates: all singleton sequences
  - <a>, <b>, <c>, <d>, <e>, <f>, <g>, <h>
- Scan database once, count support for candidates

Cand	Sup
<a>	**3**
<b>	**5**
<c>	**4**
<d>	**3**
<e>	**3**
<f>	**2**
~~<g>~~	1
~~<h>~~	1

min_sup =2

Seq. ID	Sequence
10	<(bd)cb(ac)>
20	<(bf)(ce)b(fg)>
30	<(ah)(bf)abf>
40	<(be)(ce)d>
50	<a(bd)bcb(ade)>

81

GSP: Generating Length-2 Candidates

51 length-2 Candidates

	<a>	<b>	<c>	<d>	<e>	<f>
<a>	<aa>	<ab>	<ac>	<ad>	<ae>	<af>
<b>	<ba>	<bb>	<bc>	<bd>	<be>	<bf>
<c>	<ca>	<cb>	<cc>	<cd>	<ce>	<cf>
<d>	<da>	<db>	<dc>	<dd>	<de>	<df>
<e>	<ea>	<eb>	<ec>	<ed>	<ee>	<ef>
<f>	<fa>	<fb>	<fc>	<fd>	<fe>	<ff>

	<a>	<b>	<c>	<d>	<e>	<f>
<a>		<(ab)>	<(ac)>	<(ad)>	<(ae)>	<(af)>
<b>			<(bc)>	<(bd)>	<(be)>	<(bf)>
<c>				<(cd)>	<(ce)>	<(cf)>
<d>					<(de)>	<(df)>
<e>						<(ef)>
<f>						

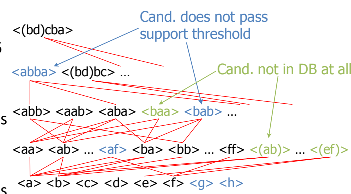
Without Apriori property, 8*8+8*7/2=92 candidates

Apriori prunes 44.57% candidates

82

The GSP Mining Process

- Scan 5: 1 candidate, 1 length-5 seq. pattern
- Scan 4: 8 candidates, 6 length-4 seq. patterns
- Scan 3: 47 candidates, 19 length-3 seq. patterns, 20 candidates not in DB at all
- Scan 2: 51 candidates, 19 length-2 seq. patterns, 10 candidates not in DB at all
- Scan 1: 8 candidates, 6 length-1 seq. patterns



min_sup =2

Seq. ID	Sequence
10	<(bd)cb(ac)>
20	<(bf)(ce)b(fg)>
30	<(ah)(bf)abf>
40	<(be)(ce)d>
50	<a(bd)bcb(ade)>

83

Candidate Generate-and-Test Drawbacks

- Huge set of candidate sequences generated
- Multiple Scans of entire database needed
  - Length of each candidate grows by one at each database scan

84

Prefix and Suffix (Projection)

- <a>, <aa>, <a(ab)> and <a(abc)> are prefixes of sequence <a(abc)(ac)d(cf)>
- Given sequence <a(abc)(ac)d(cf)>, we have:

Prefix	Suffix (Prefix-Based Projection)
<a>	<(abc)(ac)d(cf)>
<aa>	<(_bc)(ac)d(cf)>
<ab>	<(_c)(ac)d(cf)>
<bc>	<d(cf)>
<(bc)>	<(ac)d(cf)>

85

Mining Sequential Patterns by Prefix Projections

- Step 1: find length-1 frequent sequential patterns
  - <a>, <b>, <c>, <d>, <e>, <f>
- Step 2: divide search space. The complete set of sequential patterns can be partitioned into six subsets:
  - The ones having prefix <a>;
  - The ones having prefix <b>;
  - …
  - The ones having prefix <f>

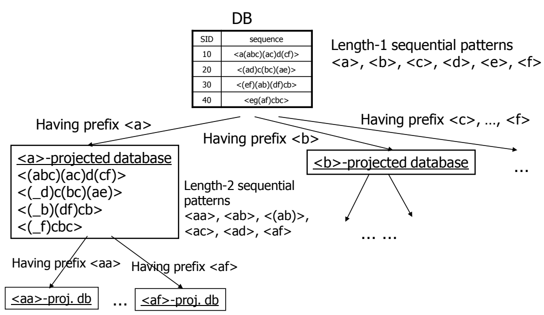
SID	sequence
10	<a(abc)(ac)d(cf)>
20	<(ad)c(bc)(ae)>
30	<(ef)(ab)(df)cb>
40	<eg(af)cbc>

Finding Seq. Patterns with Prefix <a>

- Only need to consider projections w.r.t. <a>
  - <a>-projected database: <(abc)(ac)d(cf)>, <(_d)c(bc)(ae)>, <(_b)(df)cb>, <(_f)cbc>
- Find all length-2 frequent seq. patterns having prefix <a>: <aa>, <ab>, <(ab)>, <ac>, <ad>, <af>
  - Further partition into those 6 subsets
    - Having prefix <aa>;
    - Having prefix <ab>;
    - Having prefix <(ab)>;
    - …
    - Having prefix <af>

SID	sequence
10	<a(abc)(ac)d(cf)>
20	<(ad)c(bc)(ae)>
30	<(ef)(ab)(df)cb>
40	<eg(af)cbc>

Completeness of PrefixSpan



Efficiency of PrefixSpan

- No candidate sequence needs to be generated
- Projected databases keep shrinking
- Major cost of PrefixSpan: constructing projected databases
  - Can be improved by pseudo-projections

Pseudo-Projection

- Major cost of PrefixSpan: projection
  - Postfixes of sequences often appear repeatedly in recursive projected databases
- When (projected) database can be held in memory, use pointers
  - Pointer to the sequence, offset of the postfix
- Why is this a bad idea when the (projected) database does not fit in memory?

s=<a(abc)(ac)d(cf)>

$\downarrow$ <a>

s|<a>: (, 2) <(abc)(ac)d(cf)>

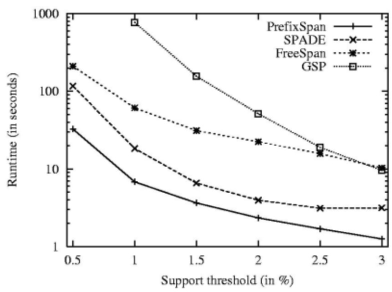
$\downarrow$ <ab>

s|<ab>: (, 4) <(_c)(ac)d(cf)>

Pseudo-Projection vs. Physical Projection

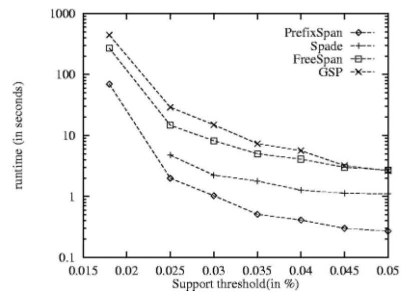
- Pseudo-projection avoids physically copying postfixes
  - Efficient in running time and space when database can be held in main memory
- Not efficient when database cannot fit in main memory
  - Disk-based random access
- Suggested Approach:
  - Integration of physical and pseudo-projection
  - Swapping to pseudo-projection when the data set fits in memory

Performance on Data Set C10T8S8I8



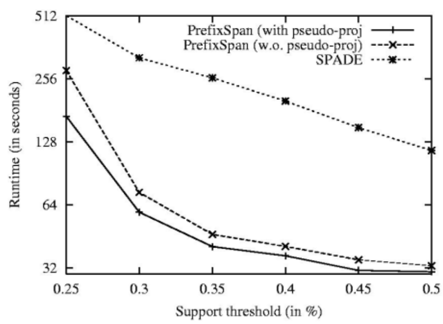
92

Performance on Data Set Gazelle



93

Effect of Pseudo-Projection



94

Sequence Mining Variations

- Multidimensional and multilevel patterns
- Constraint-based mining of sequential patterns
- Periodicity analysis
- Mining biological sequences
  - Hot research area, major topic by itself
- All these not discussed in class; see book
- Some of my own research: finding relevant sequences in bursty data; see paper

95

Frequent-Pattern Mining: Summary

- Important task in data mining
- Scalable frequent pattern mining methods
  - Apriori (itemsets, candidate generation & test)
  - GSP (sequences, candidate generation & test)
  - Projection-based (FP-growth for itemsets, PrefixSpan for sequences)
- Mining a variety of rules and interesting patterns

132

Frequent-Pattern Mining: Research Problems

- Mining fault-tolerant frequent, sequential and structured patterns
  - Patterns allows limited faults (insertion, deletion, mutation)
- Mining truly interesting patterns
  - Surprising, novel, concise,…
- Application exploration
  - E.g., DNA sequence analysis and bio-pattern classification

133