Frequent Pattern Mining
Transaction Data Analysis

- Transactions: customers’ purchases of commodities
  - \{bread, milk, cheese\} if they are bought together
- Frequent patterns: product combinations that are frequently purchased together by customers
- Frequent patterns: patterns (set of items, sequence, etc.) that occur frequently in a database [AIS93]
Why Frequent Patterns?

• What products were often purchased together?
• What are the frequent subsequent purchases after buying a iPod?
• What kinds of genes are sensitive to this new drug?
• What key-word combinations are frequently associated with web pages about game-evaluation?
Why Frequent Pattern Mining?

• Foundation for many data mining tasks
  – Association rules, correlation, causality, sequential patterns, spatial and multimedia patterns, associative classification, cluster analysis, iceberg cube, …

• Broad applications
  – Basket data analysis, cross-marketing, catalog design, sale campaign analysis, web log (click stream) analysis, …
Frequent Itemsets

- Itemset: a set of items
  - E.g., acm = \{a, c, m\}
- Support of itemsets
  - Sup(acm) = 3
- Given min_sup = 3, acm is a frequent pattern
- Frequent pattern mining: finding all frequent patterns in a database

Transaction database TDB

<table>
<thead>
<tr>
<th>TID</th>
<th>Items bought</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>f, a, c, d, g, l, m, p</td>
</tr>
<tr>
<td>200</td>
<td>a, b, c, f, l, m, o</td>
</tr>
<tr>
<td>300</td>
<td>b, f, h, j, o</td>
</tr>
<tr>
<td>400</td>
<td>b, c, k, s, p</td>
</tr>
<tr>
<td>500</td>
<td>a, f, c, e, l, p, m, n</td>
</tr>
</tbody>
</table>
A Naïve Attempt

• Generate all possible itemsets, test their supports against the database

• How to hold a large number of itemsets into main memory?
  – 100 items $\to 2^{100} - 1$ possible itemsets

• How to test the supports of a huge number of itemsets against a large database, say containing 100 million transactions?
  – A transaction of length 20 needs to update the support of $2^{20} - 1 = 1,048,575$ itemsets
Transactions in Real Applications

• A large department store often carries more than 100 thousand different kinds of items
  – Amazon.com carries more than 17,000 books relevant to data mining
• Walmart has more than 20 million transactions per day, AT&T produces more than 275 million calls per day
• Mining large transaction databases of many items is a real demand
How to Get an Efficient Method?

- Reducing the number of itemsets that need to be checked
- Checking the supports of selected itemsets efficiently
Candidate Generation & Test

• Any subset of a frequent itemset must be also frequent – an anti-monotonic property
  – A transaction containing \{beer, diaper, nuts\} also contains \{beer, diaper\}
  – \{beer, diaper, nuts\} is frequent $\rightarrow$ \{beer, diaper\} must also be frequent

• In other words, any superset of an infrequent itemset must also be infrequent
  – No superset of any infrequent itemset should be generated or tested
  – Many item combinations can be pruned!
Apriori-Based Mining

• Generate length \((k+1)\) candidate itemsets from length \(k\) frequent itemsets, and

• Test the candidates against DB
The Apriori Algorithm [AgSr94]

Data base D

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>a, c, d</td>
</tr>
<tr>
<td>20</td>
<td>b, c, e</td>
</tr>
<tr>
<td>30</td>
<td>a, b, c, e</td>
</tr>
<tr>
<td>40</td>
<td>b, e</td>
</tr>
</tbody>
</table>

Min_sup=2

Scan D

1-candidates

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Sup</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>2</td>
</tr>
<tr>
<td>b</td>
<td>3</td>
</tr>
<tr>
<td>c</td>
<td>3</td>
</tr>
<tr>
<td>d</td>
<td>1</td>
</tr>
<tr>
<td>e</td>
<td>3</td>
</tr>
</tbody>
</table>

Freq 1-itemsets

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Sup</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>2</td>
</tr>
<tr>
<td>b</td>
<td>3</td>
</tr>
<tr>
<td>c</td>
<td>3</td>
</tr>
<tr>
<td>e</td>
<td>3</td>
</tr>
</tbody>
</table>

2-candidates

<table>
<thead>
<tr>
<th>Itemset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ab</td>
</tr>
<tr>
<td>ac</td>
</tr>
<tr>
<td>ae</td>
</tr>
<tr>
<td>bc</td>
</tr>
<tr>
<td>be</td>
</tr>
<tr>
<td>ce</td>
</tr>
</tbody>
</table>

Counting

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Sup</th>
</tr>
</thead>
<tbody>
<tr>
<td>ab</td>
<td>1</td>
</tr>
<tr>
<td>ac</td>
<td>2</td>
</tr>
<tr>
<td>ae</td>
<td>1</td>
</tr>
<tr>
<td>bc</td>
<td>2</td>
</tr>
<tr>
<td>be</td>
<td>3</td>
</tr>
<tr>
<td>ce</td>
<td>2</td>
</tr>
</tbody>
</table>

Scan D

3-candidates

<table>
<thead>
<tr>
<th>Itemset</th>
</tr>
</thead>
<tbody>
<tr>
<td>bce</td>
</tr>
</tbody>
</table>

Freq 3-itemsets

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Sup</th>
</tr>
</thead>
<tbody>
<tr>
<td>bce</td>
<td>2</td>
</tr>
</tbody>
</table>

Scan D

Freq 2-itemsets

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Sup</th>
</tr>
</thead>
<tbody>
<tr>
<td>ac</td>
<td>2</td>
</tr>
<tr>
<td>bc</td>
<td>2</td>
</tr>
<tr>
<td>be</td>
<td>3</td>
</tr>
<tr>
<td>ce</td>
<td>2</td>
</tr>
</tbody>
</table>

Scan D
The Apriori Algorithm

Level-wise, candidate generation and test

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

- $L_1 = \{\text{frequent items}\}$;
- for (k = 1; $L_k \neq \emptyset$; k++) do
  - $C_{k+1}$ = 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
- return $\bigcup_k L_k$;
Important Steps of Apriori

• How to find frequent 1- and 2-itemsets?
• How to generate candidates?
  – Step 1: self-joining $L_k$
  – Step 2: pruning
• How to count supports of candidates?
Finding Frequent 1- & 2-itemsets

• Finding frequent 1-itemsets (i.e., frequent items) using a one dimensional array
  – Initialize \( c[item] = 0 \) for each item
  – For each transaction \( T \), for each item in \( T \), \( c[item]++ \);
  – If \( c[item] \geq \text{min\_sup} \), item is frequent

• Finding frequent 2-itemsets using a 2-dimensional triangle matrix
  – For items \( i, j \) (\( i < j \)), \( c[i, j] \) is the count for itemset \( ij \)
Counting Array

- A 2-dimensional triangle matrix can be implemented using a 1-dimensional array.

\[
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 \\
1 & 1 & 2 & 3 & 4 \\
2 & 5 & 6 & 7 \\
3 & 8 & 9 \\
4 & 10 \\
5 & \\
\end{array}
\]

There are n items.

For items i, j (i>j),

\[c_{i,j} = c_{(i-1)(2n-i)/2+j-i};\]

Example: \(c_{3,5} = c_{(3-1)(2*5-3)/2+5-3} = c_{9}\)
Example of Candidate-generation

- $L_3 = \{abc, abd, acd, ace, bcd\}$
- Self-joining: $L_3 \times L_3$
  - $abcd \leftarrow abc \times abd$
  - $acde \leftarrow acd \times ace$
- 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-join $L_{k-1}$
  
  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 $p.item_1$=$q.item_1$, ..., $p.item_{k-2}$=$q.item_{k-2}$, $p.item_{k-1}$ < $q.item_{k-1}$

• Step 2: pruning
  – For each itemset $c$ in $C_k$ do
    • For each $(k-1)$-subsets $s$ of $c$ do if ($s$ is not in $L_{k-1}$) then delete $c$ from $C_k$
How to Count Supports?

• 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
  – A 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
Example: Counting Supports

Subset function

Transaction: 1 2 3 5 6
Impact of Apriori

Dr. Rakesh Agrawal received the 2000 ACM SIGKDD Innovation Award
Dr. Ramakrishnan Srikant received the 2006 ACM SIGKDD Innovation Award
Apriori in SQL

- Impossible to get good performance out of pure SQL (SQL-92) based approaches alone
  - Support counting is costly
- Make use of object-relational extensions like UDFs, BLOBs, Table functions etc.
  - Get orders of magnitude improvement
- S. Sarawagi, S. Thomas, and R. Agrawal, 1998
Challenges of Freq Pat Mining

- Multiple scans of transaction database
- Huge number of candidates
- Tedious workload of support counting for candidates
Improving Apriori: Ideas

- Reducing the number of transaction database scans
- Shrinking the number of candidates
- Facilitating support counting of candidates
DIC: Reducing Number of Scans

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

Transactions

1-itemsets

2-itemsets

\[ \ldots \]

1-itemsets

2-items

3-items

DIC: Dynamic Itemset Counting

DHP: Reducing # of Candidates

• A hashing bucket count $< \text{min}_\text{sup}$ $\rightarrow$ every candidate in the bucket is infrequent
  – Candidates: $a, b, c, d, e$
  – Hash entries: $\{ab, ad, ae\} \{bd, be, de\} \ldots$
  – Large 1-itemset: $a, b, d, e$
  – The sum of counts of $\{ab, ad, ae\} < \text{min}_\text{sup}$ $\rightarrow$ ab should not be a candidate 2-itemset

• J. Park, M. Chen, and P. Yu, SIGMOD’95
  – DHP: Direct Hashing and Pruning
A 2-Scan Method by Partitioning

- Partition the database into $n$ partitions, such that each partition can be held into main memory
- Itemset $X$ is frequent $\rightarrow$ $X$ must be frequent in at least one partition
  - Scan 1: partition database and find local frequent patterns
  - Scan 2: consolidate global frequent patterns
- All local frequent itemsets can be held in main memory? A sometimes too strong assumption
- A. Savasere, E. Omiecinski, and S. Navathe, VLDB’95
Sampling for Frequent Patterns

• Select a sample of the original database, mine frequent patterns in the sample using Apriori
• Scan database once more to verify frequent itemsets found in the 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, VLDB’96
Eclat/MaxEclat and VIPER

- Tid-list: the list of transaction-ids containing an itemset
  - Vertical Data Format
- Major operation: intersections of tid-lists
- Compression of tid-lists
  - Itemset A: t1, t2 t3, \text{sup}(A)=3
  - Itemset B: t2, t3, t4, \text{sup}(B)=3
  - Itemset AB: t2, t3, \text{sup}(AB)=2

- M. Zaki et al., 1997
- P. Shenoy et al., 2000
Exercises

• What is the complexity of the frequent itemset mining problem?
  – What is the complexity of algorithm Apriori?
  – Do the improvements on Apriori discussed in this lecture improve Apriori in complexity?
Bottleneck of Freq Pattern Mining

• Multiple database scans are costly
• Mining long patterns needs many scans and generates many candidates
  – To find frequent itemset $i_1i_2\ldots i_{100}$
    • # of scans: 100
    • # of Candidates: $\binom{100}{1} + \binom{100}{2} + \cdots + \binom{100}{100} = 2^{100} - 1 \approx 1.27 \times 10^{30}$
  – Bottleneck: candidate-generation-and-test
• Can we avoid candidate generation?
Search Space of Freq. Pat. Mining

• Itemsets form a lattice
Set Enumeration Tree

- Use an order on items, enumerate itemsets in lexicographic order
  - a, ab, abc, abcd, ac, acd, ad, b, bc, bcd, bd, c, dc, d
- Reduce a lattice to a tree
Borders of Frequent Itemsets

• Frequent itemsets are connected
  – $\emptyset$ is trivially frequent
  – $X$ on the border $\rightarrow$ every subset of $X$ is frequent
Projected Databases

- To test whether Xy is frequent, we can use the X-projected database
  - The sub-database of transactions containing X
  - Check whether item y is frequent in X-projected database
Compress Database by FP-tree

- The 1st scan: find frequent items
  - Only record frequent items in FP-tree
  - F-list: f-c-a-b-m-p

- The 2nd scan: construct tree
  - Order frequent items in each transaction w.r.t. f-list
  - Explore sharing among transactions
Benefits of FP-tree

- Completeness
  - Never break a long pattern in any transaction
  - Preserve complete information for freq pattern mining
    - Not scan database anymore

- Compactness
  - Reduce irrelevant info — infrequent items are removed
  - Items in frequency descending order (f-list): the more frequently occurring, the more likely to be shared
  - Never be larger than the original database (not counting node-links and the count fields)
Partitioning Frequent Patterns

- Frequent patterns can be partitioned into subsets according to 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, or p
  - Pattern f
- Depth-first search of a set enumeration tree
  - The partitioning is complete and does not have any overlap
Find Patterns Having Item “p”

- Only transactions containing p are needed
- Form p-projected database
  - Starting at entry p of the header table
  - Follow the side-link of frequent item p
  - Accumulate all transformed prefix paths of p

$p$-projected database $\text{TDB}|_{p}$

- $f_{cam}: 2$
- $c_{b}: 1$

Local frequent item: c:3
Frequent patterns containing p
- $p: 3$, $pc: 3$
Find Pat Having Item m But No p

- Form m-projected database TDB|m
  - Item p is excluded (why?)
  - Contain fca:2, fcab:1
  - Local frequent items: f, c, a

- Build FP-tree for TDB|m
Recursive Mining

• Patterns having m but no p can be mined recursively
• Optimization: enumerate patterns from a single-branch FP-tree
  – Enumerate all combination
  – Support = that of the last item
    • m, fm, cm, am
    • fcm, fam, cam
    • fcam

\[
\text{Header table} = \begin{array}{c|c}
\text{item} & \text{support} \\
\hline
f & 3 \\
\hline
c & 3 \\
\hline
a & 3 \\
\hline
\end{array}
\]

\(m\)-projected FP-tree

\text{root} \rightarrow f:3 \rightarrow c:3 \rightarrow a:3
Enumerate Patterns From Single Prefix of FP-tree

- A (projected) FP-tree has a single prefix
  - Reduce the single prefix into one node
  - Join the mining results of the two parts

\[
\begin{align*}
\text{root} & \quad r = \quad \text{root} \\
| & | \\
a_1:n_1 & a_2:n_2 \\
| & | \\
a_3:n_3 & c_1:k_1 \\
b_1:m_1 & c_2:k_2 \\
c_3:k_3 & c_1:k_1 \\
\end{align*}
\]
FP-Growth

• Pattern-growth: recursively grow frequent patterns by pattern and database partitioning

• Algorithm
  – For each frequent item, construct its projected database, and then its projected FP-tree
  – Repeat the process on each newly created projected FP-tree
  – Until the resulted FP-tree is empty, or contains only one path – single path generates all the combinations, each of which is a frequent pattern
Scaling up by DB Projection

• What if an FP-tree cannot fit into memory?
• Database projection
  – Partition a database into a set of projected databases
  – Construct and mine FP-tree once the projected database can fit into main memory
    • Heuristic: Projected database shrinks quickly in many applications
Parallel vs. Partition Projection

- Parallel projection: form all projected database at a time
- Partition projection: propagate projections
Why Is FP-Growth Efficient?

• Divide-and-conquer strategy
  – Decompose both the mining task and DB
  – Lead to focused search of smaller databases

• Other factors
  – No candidate generation nor candidate test
  – Database compression using FP-tree
  – No repeated scan of entire database
  – Basic operations – counting local frequent items and building FP-tree, no pattern search nor pattern matching
Impact of FP-Growth

- The FP-Growth paper has been cited more than 3,000 times since 2000

Dr. Jiawei Han
Recipient of the 2004 ACM SIGKDD Innovation Award and the 2005 IEEE ICDM Research Contributions Award
Tree-Projection Method

• Find frequent 2-itemsets
• For each frequent 2-itemset $xy$, form a projected database
  – The sub-database containing $xy$
• Recursive mining
  – If $uv$ is frequent in $xy$-projected database, then $xyuv$ is a frequent pattern
Why Is Tree-Projection Fast?

• A bi-level unfolding of set enumeration tree
• Major operations
  – Finding frequent 2-itemsets: faster than matching candidates
  – Form projected databases
• Major cost: forming projections
• AAP’01
Major Costs in FP-growth

- Poor locality of FP-trees
  - Low hit rate of cache
- Building FP-trees
  - A stack of FP-trees
- Redundant information
  - Transaction abcd appears in a-, ab-, abc-, ac-, …, c- projected databases and FP-trees
Improving Locality

- Store FP-trees in pre-order depth-first traverse list

Ghoting et al., VLDB05
H-Mine

• Goal: efficient in various occasions
  – Dense vs. sparse, huge vs. memory-based data sets
• Moderate in space requirement
• Highlights
  – Effective and efficient memory-based structure and mining algorithm
  – Scalable algorithm for mining large databases by proper partitioning
  – Integration of H-mine and FP-growth
H-Structure

- Store frequent-item projections in main memory
  - Items in a transaction are sorted according to f-list
  - Each frequent item in a transaction is stored with two fields: item-id and hyper-link
  - Header table H

- Link transactions with same first item
- Scan database once

<table>
<thead>
<tr>
<th>Tid</th>
<th>Items</th>
<th>Freq-item projection</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>c, d, e, f, g, i</td>
<td>c, d, e, g</td>
</tr>
<tr>
<td>200</td>
<td>a, c, d, e, m</td>
<td>a, c, d, e</td>
</tr>
<tr>
<td>300</td>
<td>a, b, d, e, g, k</td>
<td>a, d, e, g</td>
</tr>
<tr>
<td>400</td>
<td>a, c, d, h</td>
<td>a, c, d</td>
</tr>
</tbody>
</table>

F-list = a-c-d-e-g
Find Patterns Containing Item “a”

- Only search a-projected database: transactions containing “a”
- The a-queue links all transactions in a-projected database
  – Can be traversed efficiently
Mining $a$-Projected Database

• Build $a$-header table $H_a$
• Traverse $a$-queue once, find all local frequent items within $a$-projected database
  – Local freq items: c, d, and e
  – Patterns: ac, ad and ae
• Link transactions having same next frequent item
Why Is H-Mine(Mem) Efficient?

- No candidate generation
  - It is a pattern growth method
- Search confined in a dedicated space
  - Not physically construct memory structures for projected databases
  - H-struct is for all the mining
  - Information about projected databases are collected in header tables
- No frequent patterns stored in main memory
Mining in Large Databases

- What if the H-struct is too big for memory?
- Find global frequent items
- Partition the database into n parts
  - The H-struct for each part can be held into memory
  - Mine local patterns in each part using H-mine(Mem)
    - Use relative minimum support threshold
- Consolidate global patterns in the third scan
How to Partition in H-mine?

- Partitioning in H-mine is straightforward
  - Overhead of header tables in H-mine (Mem) is small and predictable
  - Partitioning with Apriori is not easy
    - Hard to predict the space requirement of Apriori
- Global frequent items prune many local patterns in skewed partitions
Mining Dense Projected DB’s

• Challenges in dense datasets
  – Long patterns
  – Some patterns appearing in many transactions

• After projection, projected databases are denser

• Advantages of FP-tree
  – Compress dense databases many times
  – No candidate generation
  – Sub-patterns can be enumerated from long patterns

• Build FP-tree for dense projected databases
  – Empirical switching point: 1%
Advantages of H-Mine

• Have very small space overhead
• Absorb nice features of FP-growth
• Create no physical projected database
• Watch the density of projected databases, turn to FP-growth when necessary
• Propose space-preserving mining
  – Scalable in very large database
  – Feasible even with very small memory
  – Go beyond frequent pattern mining
Further Developments

• OP – opportunistic projection (LPWH02)
  – Opportunistically choose between array-based and tree-based representations of projected databases

• Diffsets for vertical mining (ZaGo03)
  – Only record the differences in the tids of a candidate pattern from its generating frequent patterns
Mining and Storing Patterns on Disk

- The (projected) databases can be huge
- Inverted matrix (EiZa03)
  - Indexed transaction database on disk
- CFP-tree – condensed FP-tree (LLLY03)
  - Organizing frequent patterns on disk
  - Efficient query answering algorithm
Multiple-level Association Rules

- Items often form hierarchy
- Flexible support settings: Items at the lower level are expected to have lower support.
- Transaction database can be encoded based on dimensions and levels
- Explore shared multi-level mining

uniform support

<table>
<thead>
<tr>
<th>Level 1</th>
<th>min_sup = 5%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Milk</td>
<td>[support = 10%]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Level 2</th>
<th>min_sup = 5%</th>
</tr>
</thead>
<tbody>
<tr>
<td>2% Milk</td>
<td>[support = 6%]</td>
</tr>
<tr>
<td>Skim Milk</td>
<td>[support = 4%]</td>
</tr>
</tbody>
</table>

reduced support

<table>
<thead>
<tr>
<th>Level 1</th>
<th>min_sup = 5%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Milk</td>
<td>[support = 10%]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Level 2</th>
<th>min_sup = 3%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skim Milk</td>
<td>[support = 4%]</td>
</tr>
</tbody>
</table>
Multi-dimensional Rules

- **Single-dimensional rules:**
  - $\text{bears}(X, \text{“milk”}) \Rightarrow \text{bears}(X, \text{“bread”})$

- **MD rules: $\geq 2$ dimensions or predicates**
  - Inter-dimension assoc. rules (no repeated predicates)
    - $\text{age}(X, \text{“19-25”}) \land \text{occupation}(X, \text{“student”}) \Rightarrow \text{bears}(X, \text{“coke”})$
  - Hybrid-dimension assoc. rules (repeated predicates)
    - $\text{age}(X, \text{“19-25”}) \land \text{bears}(X, \text{“popcorn”}) \Rightarrow \text{bears}(X, \text{“coke”})$

- **Categorical Attributes:** finite number of possible values, no order among values

- **Quantitative Attributes:** numeric, implicit order
Quantitative Association Rules

Numeric attributes are *dynamically* discretized to maximize the confidence or compactness of the rules.

2-D quantitative association rules: $A_{\text{quan1}} \land A_{\text{quan2}} \Rightarrow A_{\text{cat}}$

Cluster “adjacent” association rules to form general rules using a 2-D grid.

$$\text{age}(X, "33-34") \land \text{income}(X, "30K - 50K") \Rightarrow \text{buys}(X, "high resolution TV")$$
Distance-based Rules

- Binning methods do not capture semantics of interval data
- Distance-based partitioning
  - Density/number of points in an interval
  - “Closure” of points in an interval

<table>
<thead>
<tr>
<th>Price</th>
<th>Equi-width</th>
<th>Equi-depth</th>
<th>Distance-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>[0,10]</td>
<td>[7,20]</td>
<td>[7,7]</td>
</tr>
<tr>
<td>20</td>
<td>[11,20]</td>
<td>[22,50]</td>
<td>[20,22]</td>
</tr>
<tr>
<td>22</td>
<td>[21,30]</td>
<td>[51,53]</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>[31,40]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>[41,50]</td>
<td></td>
<td>[50,53]</td>
</tr>
<tr>
<td>53</td>
<td>[51,60]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Flexible Support Constraints

• Why flexible support constraints?
  – Real life occurrence frequencies vary greatly
    • Diamond, watch, pens in a shopping basket
  – Uniform support may not be an interesting model
• A flexible model
  – The lower-level, the more dimension combination, and the long pattern length, usually the smaller support
  – General rules should be easy to specify and understand
  – Special items and special group of items should be specified individually and have higher priority
  – Passing (support) thresholds vs. printing thresholds
Misleading Rules

• Play basketball → eat cereal [40%, 66.7%]
  – The overall percentage of students eating cereal is 75%, is higher than 66.7%
  – Play basketball → not eat cereal [20%, 33.3%] is more accurate, though with lower support and confidence

<table>
<thead>
<tr>
<th></th>
<th>Basketball</th>
<th>Not basketball</th>
<th>Sum (row)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cereal</td>
<td>2000</td>
<td>1750</td>
<td>3750</td>
</tr>
<tr>
<td>Not cereal</td>
<td>1000</td>
<td>250</td>
<td>1250</td>
</tr>
<tr>
<td>Sum(col.)</td>
<td>3000</td>
<td>2000</td>
<td>5000</td>
</tr>
</tbody>
</table>
Correlation and Lift

- \( P(B|A)/P(B) \) is called the lift of rule \( A \rightarrow B \)

\[
corr_{A,B} = \frac{P(A \cup B)}{P(A)P(B)}
\]

- Play basketball \( \rightarrow \) eat cereal (lift: 0.89)
- Play basketball \( \rightarrow \) not eat cereal (lift: 1.33)

<table>
<thead>
<tr>
<th></th>
<th>Basketball</th>
<th>Not basketball</th>
<th>Sum (row)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cereal</td>
<td>2000</td>
<td>1750</td>
<td>3750</td>
</tr>
<tr>
<td>Not cereal</td>
<td>1000</td>
<td>250</td>
<td>1250</td>
</tr>
<tr>
<td>Sum(col.)</td>
<td>3000</td>
<td>2000</td>
<td>5000</td>
</tr>
</tbody>
</table>
Lift and $\chi^2$ Good for Correlation?

- “Buy walnuts $\Rightarrow$ buy milk [1%, 80%]” is misleading
  - if 85% of customers buy milk

\[
all\_conf = \frac{\text{sup}(X)}{\text{max}\_item\_sup(X)}
\]

\[
coh = \frac{\text{sup}(X)}{|universe(X)|}
\]

<table>
<thead>
<tr>
<th>DB</th>
<th>m, c</th>
<th>$\neg$m, c</th>
<th>m$&amp;$c</th>
<th>$\neg$m$&amp;$c</th>
<th>lift</th>
<th>all-conf</th>
<th>coh</th>
<th>$\chi^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>1000</td>
<td>100</td>
<td>100</td>
<td>10,000</td>
<td>9.26</td>
<td>0.91</td>
<td>0.83</td>
<td>9055</td>
</tr>
<tr>
<td>A2</td>
<td>100</td>
<td>1000</td>
<td>1000</td>
<td>100,000</td>
<td>8.44</td>
<td>0.09</td>
<td>0.05</td>
<td>670</td>
</tr>
<tr>
<td>A3</td>
<td>1000</td>
<td>100</td>
<td>10000</td>
<td>100,000</td>
<td>9.18</td>
<td>0.09</td>
<td>0.09</td>
<td>8172</td>
</tr>
<tr>
<td>A4</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>1.00</td>
<td>0.50</td>
<td>0.33</td>
<td>0</td>
</tr>
</tbody>
</table>
## Correlation Measures

<table>
<thead>
<tr>
<th>symbol</th>
<th>measure</th>
<th>range</th>
<th>formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\phi$</td>
<td>$\phi$-coefficient</td>
<td>$-1 \ldots 1$</td>
<td>$\sqrt{P(A)P(B)(1-P(A))(1-P(B))}$</td>
</tr>
<tr>
<td>$Q$</td>
<td>Yule’s Q</td>
<td>$-1 \ldots 1$</td>
<td>$P(A,B)P(\overline{A},\overline{B})-P(A,\overline{B})P(A,B)$</td>
</tr>
<tr>
<td>$Y$</td>
<td>Yule’s Y</td>
<td>$-1 \ldots 1$</td>
<td>$P(A,B)P(\overline{A},\overline{B})-P(A,B)P(A,\overline{B})$</td>
</tr>
<tr>
<td>$k$</td>
<td>Cohen’s</td>
<td>$-1 \ldots 1$</td>
<td>$\sqrt{P(A,B)P(\overline{A},\overline{B})}+\sqrt{P(A,B)P(A,\overline{B})}$</td>
</tr>
<tr>
<td>$PS$</td>
<td>Piatetsky-Shapiro’s</td>
<td>$-0.25 \ldots 0.25$</td>
<td>$P(A,B)-P(A)P(B)$</td>
</tr>
<tr>
<td>$F$</td>
<td>Certainty factor</td>
<td>$-1 \ldots 1$</td>
<td>$\max(\frac{P(B</td>
</tr>
<tr>
<td>$AV$</td>
<td>added value</td>
<td>$-0.5 \ldots 1$</td>
<td>$\max(P(B</td>
</tr>
<tr>
<td>$K$</td>
<td>Klosgen’s Q</td>
<td>$-0.33 \ldots 0.38$</td>
<td>$\sqrt{P(A,B)\max(P(B</td>
</tr>
<tr>
<td>$g$</td>
<td>Goodman-kruskal’s</td>
<td>$0 \ldots 1$</td>
<td>$\frac{\sum_j \max_k P(A_j,B_k)+\sum_k \max_j P(A_j,B_k)-\max_j P(A_j)\max_k P(B_k)}{2^{\max_j P(A_j)-\max_k P(B_k)}}$</td>
</tr>
<tr>
<td>$M$</td>
<td>Mutual Information</td>
<td>$0 \ldots 1$</td>
<td>$\min(\sum_i P(A_i), \log P(A_i), \sum_i P(B_i), \log P(B_i))$</td>
</tr>
<tr>
<td>$J$</td>
<td>J-Measure</td>
<td>$0 \ldots 1$</td>
<td>$\max(P(A,B)\log\frac{P(B</td>
</tr>
<tr>
<td>$G$</td>
<td>Gini index</td>
<td>$0 \ldots 1$</td>
<td>$\max(P(A)</td>
</tr>
<tr>
<td>$s$</td>
<td>support</td>
<td>$0 \ldots 1$</td>
<td>$\max(\frac{NP(A,B)+1}{NP(A)+2}, \frac{NP(A,B)+1}{NP(B)+2})$</td>
</tr>
<tr>
<td>$c$</td>
<td>confidence</td>
<td>$0 \ldots 1$</td>
<td>$\max(\frac{NP(A,B)+1}{NP(A)+2}, \frac{NP(A,B)+1}{NP(B)+2})$</td>
</tr>
<tr>
<td>$L$</td>
<td>Laplace</td>
<td>$0 \ldots 1$</td>
<td>$\frac{1}{P(A,B)}\sqrt{P(A)P(B)}$</td>
</tr>
<tr>
<td>$IS$</td>
<td>Cosine</td>
<td>$0 \ldots 1$</td>
<td>$\frac{P(A,B)^2}{P(A)+P(B)-P(A,B)}$</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>coherence (Jaccard)</td>
<td>$0 \ldots 1$</td>
<td>$\max(P(A)+P(B)-P(A,B))$</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>all confidence</td>
<td>$0 \ldots 1$</td>
<td>$\max(P(A,B)\log\frac{P(\overline{A})}{P(A)})$</td>
</tr>
<tr>
<td>$o$</td>
<td>odds ratio</td>
<td>$0 \ldots \infty$</td>
<td>$\max(P(A,B)\log\frac{P(\overline{A})}{P(A)})$</td>
</tr>
<tr>
<td>$V$</td>
<td>Conviction</td>
<td>$0.5 \ldots \infty$</td>
<td>$\max(\frac{P(A)P(B)}{P(\overline{A})}, \frac{P(B)P(\overline{A})}{P(B)})$</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>lift</td>
<td>$0 \ldots \infty$</td>
<td>$\frac{P(A,B)}{P(\overline{A})}$</td>
</tr>
<tr>
<td>$S$</td>
<td>Collective strength</td>
<td>$0 \ldots \infty$</td>
<td>$\frac{P(A,B)+P(\overline{A})\overline{B}}{P(\overline{A})}$</td>
</tr>
<tr>
<td>$\chi^2$</td>
<td>$\chi^2$</td>
<td>$0 \ldots \infty$</td>
<td>$\sum_i \frac{(P(A_i)\overline{E})^2}{E_i}$</td>
</tr>
</tbody>
</table>
All-conf and Coherence

- Lift and $\chi^2$ are not good measures for correlations in large transactional databases
- All-conf or coherence could be good measures (Omiecinski, TKDE’03)
- Both all-conf and coherence have the downward closure property
- Efficient algorithms can be derived for mining (Lee et al., ICDM’03)
Effectiveness of Freq Pat Mining

• Too many patterns!
  – A pattern $a_1a_2\ldots a_n$ contains $2^n-1$ subpatterns
  – Understanding many patterns is difficult or even impossible for human being

• Non-focused mining
  – A manager may be only interested in patterns involving some items (s)he manages
  – A user is often interested in patterns satisfying some constraints
Itemset Lattice

<table>
<thead>
<tr>
<th>Tid</th>
<th>transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>ABD</td>
</tr>
<tr>
<td>20</td>
<td>ABC</td>
</tr>
<tr>
<td>30</td>
<td>AD</td>
</tr>
<tr>
<td>40</td>
<td>ABCD</td>
</tr>
<tr>
<td>50</td>
<td>CD</td>
</tr>
</tbody>
</table>

Min_sup=2

<table>
<thead>
<tr>
<th>Length</th>
<th>Frequent itemsets</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A, B, C, D</td>
</tr>
<tr>
<td>2</td>
<td>AB, AC, AD, BC, BD, CD</td>
</tr>
<tr>
<td>3</td>
<td>ABC, ABD, ACD</td>
</tr>
</tbody>
</table>
Max-Patterns and Border

<table>
<thead>
<tr>
<th>Tid</th>
<th>transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>ABD</td>
</tr>
<tr>
<td>20</td>
<td>ABC</td>
</tr>
<tr>
<td>30</td>
<td>AD</td>
</tr>
<tr>
<td>40</td>
<td>ABCD</td>
</tr>
<tr>
<td>50</td>
<td>CD</td>
</tr>
</tbody>
</table>

Min_sup=2

<table>
<thead>
<tr>
<th>Length</th>
<th>Frequent itemsets</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A, B, C, D</td>
</tr>
<tr>
<td>2</td>
<td>AB, AC, AD, BC, BD, CD</td>
</tr>
<tr>
<td>3</td>
<td>ABC, ABD</td>
</tr>
</tbody>
</table>
Borders and Max-patterns

- Max-patterns: borders of frequent patterns
  - Any subset of max-pattern is frequent
  - Any superset of max-pattern is infrequent
  - Cannot generate rules
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,
- Since BCDE is a max-pattern, no need to check BCD, BDE, CDE in later scan
- Bayardo, SIGMOD’98

<table>
<thead>
<tr>
<th>Tid</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>A,B,C,D,E</td>
</tr>
<tr>
<td>20</td>
<td>B,C,D,E,</td>
</tr>
<tr>
<td>30</td>
<td>A,C,D,F</td>
</tr>
</tbody>
</table>

Min_sup=2

Potential max-patterns
Patterns and Support Counts

Tid | transaction
---|---
10  | ABD
20  | ABC
30  | AD
40  | ABCD
50  | CD

Min_sup=2

Len | Frequent itemsets
---|---
1  | A:4, B:4, C:3, D:4
2  | AB:3, AC:2, AD:3, BC:3, BD:2, CD:2
3  | ABC:2, ABD:2
Frequent Closed Patterns

- For frequent itemset $X$, if there exists no item $y$ not in $X$ s.t. every transaction containing $X$ also contains $y$, then $X$ is a frequent closed pattern
  - “acdf” is a frequent closed pattern
- Concise rep. of freq pats
  - Can generate non-redundant rules
- Reduce # of patterns and rules
- N. Pasquier et al. In ICDT’99

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>a, c, d, e, f</td>
</tr>
<tr>
<td>20</td>
<td>a, b, e</td>
</tr>
<tr>
<td>30</td>
<td>c, e, f</td>
</tr>
<tr>
<td>40</td>
<td>a, c, d, f</td>
</tr>
<tr>
<td>50</td>
<td>c, e, f</td>
</tr>
</tbody>
</table>
CLOSET for Frequent Closed Patterns

- **Flist**: list of all freq items in support asc. 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

- **PHM’00**

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>a, c, d, e, f</td>
</tr>
<tr>
<td>20</td>
<td>a, b, e</td>
</tr>
<tr>
<td>30</td>
<td>c, e, f</td>
</tr>
<tr>
<td>40</td>
<td>a, c, d, f</td>
</tr>
<tr>
<td>50</td>
<td>c, e, f</td>
</tr>
</tbody>
</table>
The CHARM Method

• Use vertical data format: \( t(AB) = \{T1, T12, \ldots \} \)
• Derive closed pattern based on vertical intersections
  – \( t(X) = t(Y) \): \( X \) and \( Y \) always happen together
  – \( t(X) \subseteq t(Y) \): transaction having \( X \) always has \( Y \)
• Use diffset to accelerate mining
  – Only keep track of difference of tids
  – \( t(X) = \{T1, T2, T3\}, \ t(Xy) = \{T1, T3\} \)
  – Diffset\( (Xy, X) = \{T2\} \)
Closed and Max-patterns

- Closed pattern mining algorithms can be adapted to mine max-patterns
  - A max-pattern must be closed
- Depth-first search methods have advantages over breadth-first search ones
  - Why?
Condensed Freq Pattern Base

• Practical observation: in many applications, a good approximation on support count could be good enough
  – Support=10000 → Support in range 10000 ± 1%

• Making frequent pattern mining more realistic
  – A small deviation has a minor effect on analysis
  – Condensed FP-base leads to more effective mining
  – Computing a condensed FP-base may lead to more efficient mining
Condensed FP-base Mining

• Compute a condensed FP-base with a guaranteed maximal error bound.
• Given: a transaction database, a user-specified support threshold, and a user-specified error bound
• Find a subset of frequent patterns & a function
  – Determine whether a pattern is frequent
  – Determine the support range
• Pei et al. ICDM’02
An Example

Support threshold: \( \text{min\_sup} = 1 \)

Error bound: \( k = 2 \)

<table>
<thead>
<tr>
<th>Transaction-id</th>
<th>Itemset</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>a</td>
</tr>
<tr>
<td>20</td>
<td>ab</td>
</tr>
<tr>
<td>30</td>
<td>abc</td>
</tr>
<tr>
<td>40</td>
<td>abcd</td>
</tr>
<tr>
<td>50</td>
<td>cd</td>
</tr>
<tr>
<td>60</td>
<td>abd</td>
</tr>
<tr>
<td>70</td>
<td>bcd</td>
</tr>
</tbody>
</table>
Another Base

Support threshold: min_sup = 1
Error bound: k = 2
Approximation Functions

• NOT unique
  – Different condensed FP-bases have different approximation function

• Optimization on space requirement
  – The less space required, the better compression effect
  – compression ratio

\[ \delta = \frac{\text{# of patterns in the condensed FP-base}}{\text{total # of frequent patterns}} \]
Constraint-based Data Mining

• Find all the patterns in a database autonomously?
  – The patterns could be too many but not focused!

• Data mining should be interactive
  – User directs what to be mined

• Constraint-based mining
  – User flexibility: provides constraints on what to be mined
  – System optimization: push constraints for efficient mining
Constraints in Data Mining

- Knowledge type constraint
  - classification, association, etc.
- Data constraint — using SQL-like queries
  - find product pairs sold together in stores in New York
- Dimension/level constraint
  - in relevance to region, price, brand, customer category
- Rule (or pattern) constraint
  - small sales (price < $10) triggers big sales (sum >$200)
- Interestingness constraint
  - strong rules: support and confidence
Constrained Mining vs. Search

• Constrained mining vs. constraint-based search
  – Both aim at reducing search space
  – Finding all patterns vs. some (or one) answers satisfying constraints
  – Constraint-pushing vs. heuristic search
  – An interesting research problem on integrating both

• Constrained mining vs. DBMS query processing
  – Database query processing requires to find all
  – Constrained pattern mining shares a similar philosophy as pushing selections deeply in query processing
Optimization

• Mining frequent patterns with constraint C
  – Sound: only find patterns satisfying the constraints C
  – Complete: find all patterns satisfying the constraints C

• A naïve solution
  – Constraint test as a post-processing

• More efficient approaches
  – Analyze the properties of constraints
  – Push constraints as deeply as possible into frequent pattern mining
Anti-Monotonicity

- Anti-monotonicity
  - An itemset $S$ violates the constraint, so does any of its superset
  - $\sum(S.\text{Price}) \leq v$ is anti-monotone
  - $\sum(S.\text{Price}) \geq v$ is not anti-monotone

- Example
  - $C$: $\text{range}(S.\text{profit}) \leq 15$
  - Itemset $ab$ violates $C$
  - So does every superset of $ab$

<table>
<thead>
<tr>
<th>TID</th>
<th>Transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>a, b, c, d, f</td>
</tr>
<tr>
<td>20</td>
<td>b, c, d, f, g, h</td>
</tr>
<tr>
<td>30</td>
<td>a, c, d, e, f</td>
</tr>
<tr>
<td>40</td>
<td>c, e, f, g</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Item</th>
<th>Profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>40</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
</tr>
<tr>
<td>c</td>
<td>-20</td>
</tr>
<tr>
<td>d</td>
<td>10</td>
</tr>
<tr>
<td>e</td>
<td>-30</td>
</tr>
<tr>
<td>f</td>
<td>30</td>
</tr>
<tr>
<td>g</td>
<td>20</td>
</tr>
<tr>
<td>h</td>
<td>-10</td>
</tr>
</tbody>
</table>
## Anti-monotonic Constraints

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Antimotone</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v \in S$</td>
<td>No</td>
</tr>
<tr>
<td>$S \supseteq V$</td>
<td>no</td>
</tr>
<tr>
<td>$S \subseteq V$</td>
<td>yes</td>
</tr>
<tr>
<td>$\min(S) \leq v$</td>
<td>no</td>
</tr>
<tr>
<td>$\min(S) \geq v$</td>
<td>yes</td>
</tr>
<tr>
<td>$\max(S) \leq v$</td>
<td>yes</td>
</tr>
<tr>
<td>$\max(S) \geq v$</td>
<td>no</td>
</tr>
<tr>
<td>$\text{count}(S) \leq v$</td>
<td>yes</td>
</tr>
<tr>
<td>$\text{count}(S) \geq v$</td>
<td>no</td>
</tr>
<tr>
<td>$\sum(S) \leq v \ (a \in S, a \geq 0)$</td>
<td>yes</td>
</tr>
<tr>
<td>$\sum(S) \geq v \ (a \in S, a \geq 0)$</td>
<td>no</td>
</tr>
<tr>
<td>$\text{range}(S) \leq v$</td>
<td>yes</td>
</tr>
<tr>
<td>$\text{range}(S) \geq v$</td>
<td>no</td>
</tr>
<tr>
<td>$\text{avg}(S) \theta v, \theta \in {=, \leq, \geq}$</td>
<td>convertible</td>
</tr>
<tr>
<td>$\text{support}(S) \geq \xi$</td>
<td>yes</td>
</tr>
<tr>
<td>$\text{support}(S) \leq \xi$</td>
<td>no</td>
</tr>
</tbody>
</table>
Monotonicity

• Monotonicity
  – An itemset $S$ satisfies the constraint, so does any of its superset
  – $\sum(S.Price) \geq v$ is monotone
  – $\min(S.Price) \leq v$ is monotone

• Example
  – $C$: range($S.profit) \geq 15$
  – Itemset $ab$ satisfies $C$
  – So does every superset of $ab$
## Monotonic Constraints

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Monotone</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v \in S$</td>
<td>yes</td>
</tr>
<tr>
<td>$S \supseteq V$</td>
<td>yes</td>
</tr>
<tr>
<td>$S \subseteq V$</td>
<td>no</td>
</tr>
<tr>
<td>$\min(S) \leq v$</td>
<td>yes</td>
</tr>
<tr>
<td>$\min(S) \geq v$</td>
<td>no</td>
</tr>
<tr>
<td>$\max(S) \leq v$</td>
<td>no</td>
</tr>
<tr>
<td>$\max(S) \geq v$</td>
<td>yes</td>
</tr>
<tr>
<td>$\text{count}(S) \leq v$</td>
<td>no</td>
</tr>
<tr>
<td>$\text{count}(S) \geq v$</td>
<td>yes</td>
</tr>
<tr>
<td>$\text{sum}(S) \leq v \ (a \in S, a \geq 0)$</td>
<td>no</td>
</tr>
<tr>
<td>$\text{sum}(S) \geq v \ (a \in S, a \geq 0)$</td>
<td>yes</td>
</tr>
<tr>
<td>$\text{range}(S) \leq v$</td>
<td>no</td>
</tr>
<tr>
<td>$\text{range}(S) \geq v$</td>
<td>yes</td>
</tr>
<tr>
<td>$\theta \in {=, \leq, \geq}$</td>
<td>convertible</td>
</tr>
<tr>
<td>$\text{support}(S) \geq \xi$</td>
<td>no</td>
</tr>
<tr>
<td>$\text{support}(S) \leq \xi$</td>
<td>yes</td>
</tr>
</tbody>
</table>
Succinctness

• Succinctness:
  – Without looking at the transaction database, whether an itemset S satisfies constraint C can be determined based on the selection of items
  – \( \min(S.\text{Price}) \leq v \) is succinct
  – \( \sum(S.\text{Price}) \geq v \) is not succinct

• Optimization: If C is succinct, C is pre-counting pushable
  – More details in the textbook
## Succinct Constraints

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Succinct</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v \in S$</td>
<td>yes</td>
</tr>
<tr>
<td>$S \supseteq V$</td>
<td>yes</td>
</tr>
<tr>
<td>$S \subseteq V$</td>
<td>yes</td>
</tr>
<tr>
<td>$\min(S) \leq v$</td>
<td>yes</td>
</tr>
<tr>
<td>$\min(S) \geq v$</td>
<td>yes</td>
</tr>
<tr>
<td>$\max(S) \leq v$</td>
<td>yes</td>
</tr>
<tr>
<td>$\max(S) \geq v$</td>
<td>yes</td>
</tr>
<tr>
<td>$\text{count}(S) \leq v$</td>
<td>weakly</td>
</tr>
<tr>
<td>$\text{count}(S) \geq v$</td>
<td>weakly</td>
</tr>
<tr>
<td>$\text{sum}(S) \leq v$</td>
<td>no</td>
</tr>
<tr>
<td>$\text{sum}(S) \geq v$</td>
<td>no</td>
</tr>
<tr>
<td>$\text{range}(S) \leq v$</td>
<td>no</td>
</tr>
<tr>
<td>$\text{range}(S) \geq v$</td>
<td>no</td>
</tr>
<tr>
<td>$\text{avg}(S) \theta v$, $\theta \in {=, \leq, \geq}$</td>
<td>no</td>
</tr>
<tr>
<td>$\text{support}(S) \geq \xi$</td>
<td>no</td>
</tr>
<tr>
<td>$\text{support}(S) \leq \xi$</td>
<td>no</td>
</tr>
</tbody>
</table>
Converting “Tough” Constraints

- Convert tough constraints into anti-monotone or monotone by properly ordering items
- Examine C: \( \text{avg}(S.\text{profit}) \geq 25 \)
  - Order items in value-descending order
    - \(<a, f, g, d, b, h, c, e>\)
  - If an itemset \(afb\) violates C
    - So does \(afbh, afb^*\)
    - It becomes anti-monotone!

<table>
<thead>
<tr>
<th>TID</th>
<th>Transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>a, b, c, d, f</td>
</tr>
<tr>
<td>20</td>
<td>b, c, d, f, g, h</td>
</tr>
<tr>
<td>30</td>
<td>a, c, d, e, f</td>
</tr>
<tr>
<td>40</td>
<td>c, e, f, g</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Item</th>
<th>Profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>40</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
</tr>
<tr>
<td>c</td>
<td>-20</td>
</tr>
<tr>
<td>d</td>
<td>10</td>
</tr>
<tr>
<td>e</td>
<td>-30</td>
</tr>
<tr>
<td>f</td>
<td>30</td>
</tr>
<tr>
<td>g</td>
<td>20</td>
</tr>
<tr>
<td>h</td>
<td>-10</td>
</tr>
</tbody>
</table>
Convertible Constraints

- Let R be an order of items
- Convertible anti-monotone
  - If an itemset S violates a constraint C, so does every itemset having S as a prefix w.r.t. R
  - Ex. $\text{avg}(S) \leq v$ w.r.t. item value descending order
- Convertible monotone
  - If an itemset S satisfies constraint C, so does every itemset having S as a prefix w.r.t. R
  - Ex. $\text{avg}(S) \geq v$ w.r.t. item value descending order
StronglyConvertibleConstraints

- **avg(X) ≥ 25** is convertible anti-monotone w.r.t. item value descending order R: <a, f, g, d, b, h, c, e>
  - Itemset af violates a constraint C, so does every itemset with af as prefix, such as afd
- **avg(X) ≥ 25** is convertible monotone w.r.t. item value ascending order R⁻¹: <e, c, h, b, d, g, f, a>
  - Itemset d satisfies a constraint C, so does itemsets df and dfa, which having d as a prefix
- Thus, **avg(X) ≥ 25** is strongly convertible

<table>
<thead>
<tr>
<th>Item</th>
<th>Profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>40</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
</tr>
<tr>
<td>c</td>
<td>-20</td>
</tr>
<tr>
<td>d</td>
<td>10</td>
</tr>
<tr>
<td>e</td>
<td>-30</td>
</tr>
<tr>
<td>f</td>
<td>30</td>
</tr>
<tr>
<td>g</td>
<td>20</td>
</tr>
<tr>
<td>h</td>
<td>-10</td>
</tr>
</tbody>
</table>
## Convertible Constraints

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Convertible anti-monotone</th>
<th>Convertible monotone</th>
<th>Strongly convertible</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{avg}(S) \leq, \geq v$</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>$\text{median}(S) \leq, \geq v$</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>$\text{sum}(S) \leq v$ (items could be of any value, $v \geq 0$)</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>$\text{sum}(S) \leq v$ (items could be of any value, $v \leq 0$)</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>$\text{sum}(S) \geq v$ (items could be of any value, $v \geq 0$)</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>$\text{sum}(S) \geq v$ (items could be of any value, $v \leq 0$)</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
## A General Picture of Constraints

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Antimotone</th>
<th>Monotone</th>
<th>Succinct</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v \in S$</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>$S \supseteq V$</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>$S \subseteq V$</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>$\min(S) \leq v$</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>$\min(S) \geq v$</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>$\max(S) \leq v$</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>$\max(S) \geq v$</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>$\text{count}(S) \leq v$</td>
<td>yes</td>
<td>no</td>
<td>weakly</td>
</tr>
<tr>
<td>$\text{count}(S) \geq v$</td>
<td>no</td>
<td>yes</td>
<td>weakly</td>
</tr>
<tr>
<td>$\text{sum}(S) \leq v \ (a \in S, a \geq 0)$</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>$\text{sum}(S) \geq v \ (a \in S, a \geq 0)$</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>$\text{range}(S) \leq v$</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>$\text{range}(S) \geq v$</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>$\text{avg}(S) \theta v, \theta \in { =, \leq, \geq }$</td>
<td>convertible</td>
<td>convertible</td>
<td>no</td>
</tr>
<tr>
<td>$\text{support}(S) \geq \xi$</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>$\text{support}(S) \leq \xi$</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
</tbody>
</table>
Classification of Constraints

Inconvertible

Convertible anti-monotone

Succinct

Convertible monotone

Strongly convertible

Convertible

Antimonotone

Monotone
Can Apriori Handle Convertible Constraint?

• A convertible, not monotone nor anti-monotone nor succinct constraint cannot be pushed deep into the an Apriori mining algorithm
  – Within the level wise framework, no direct pruning based on the constraint can be made
  – Itemset df violates constraint C: \( \text{avg}(X) \geq 25 \)
  – Since adf satisfies C, Apriori needs df to assemble adf, df cannot be pruned

• But it can be pushed into frequent-pattern growth framework!

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>40</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
</tr>
<tr>
<td>c</td>
<td>-20</td>
</tr>
<tr>
<td>d</td>
<td>10</td>
</tr>
<tr>
<td>e</td>
<td>-30</td>
</tr>
<tr>
<td>f</td>
<td>30</td>
</tr>
<tr>
<td>g</td>
<td>20</td>
</tr>
<tr>
<td>h</td>
<td>-10</td>
</tr>
</tbody>
</table>
Mining With Convertible Constraints

• C: \( \text{avg}(S.\text{profit}) \geq 25 \)
• List of items in every transaction in value descending order \( R \):
  – \(<a, f, g, d, b, h, c, e>\)
  – C is convertible anti-monotone w.r.t. \( R \)
• Scan transaction DB once
  – remove infrequent items
    • Item h in transaction 40 is dropped
  – Itemsets a and f are good

<table>
<thead>
<tr>
<th>TID</th>
<th>Transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>a, f, d, b, c</td>
</tr>
<tr>
<td>20</td>
<td>f, g, d, b, c</td>
</tr>
<tr>
<td>30</td>
<td>a, f, d, c, e</td>
</tr>
<tr>
<td>40</td>
<td>f, g, h, c, e</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Item</th>
<th>Profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>40</td>
</tr>
<tr>
<td>f</td>
<td>30</td>
</tr>
<tr>
<td>g</td>
<td>20</td>
</tr>
<tr>
<td>d</td>
<td>10</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
</tr>
<tr>
<td>h</td>
<td>-10</td>
</tr>
<tr>
<td>c</td>
<td>-20</td>
</tr>
<tr>
<td>e</td>
<td>-30</td>
</tr>
</tbody>
</table>
From Itemsets to Sequences

- Itemsets: combinations of items, no temporal order
- Temporal order is important in many situations
  - Time-series databases and sequence databases
  - Frequent patterns $\rightarrow$ (frequent) sequential patterns
- Applications of sequential pattern mining
  - Customer shopping sequences:
    - First buy computer, then iPod, and then digital camera, within 3 months.
  - Medical treatment, natural disasters, science and engineering processes, stocks and markets, telephone calling patterns, Web log clickthrough streams, DNA sequences and gene structures
What Is Sequential Pattern Mining?

• Given a set of sequences, find the complete set of frequent subsequences

A sequence database

<table>
<thead>
<tr>
<th>SID</th>
<th>sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>&lt;a(abc)(ac)d(cf)&gt;</td>
</tr>
<tr>
<td>20</td>
<td>&lt;(ad)c(bc)(ae)&gt;</td>
</tr>
<tr>
<td>30</td>
<td>&lt;(ef)(ab)(df)cb&gt;</td>
</tr>
<tr>
<td>40</td>
<td>&lt;eg(af)cbc&gt;</td>
</tr>
</tbody>
</table>

A sequence: <(ef)(ab)(df)c b>  
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 \( \text{min}_\text{sup} = 2 \), <(ab)c> is a sequential pattern
Challenges in Seq Pat Mining

• A huge number of possible sequential patterns are hidden in databases

• A mining algorithm should
  – Find the complete set of patterns satisfying the minimum support (frequency) threshold
  – Be highly efficient, scalable, involving only a small number of database scans
  – Be able to incorporate various kinds of user-specific constraints
Apriori Property of Seq Patterns

- Apriori property in sequential patterns
  - If a sequence S is infrequent, then none of the super-sequences of S is frequent
  - E.g, <hb> is infrequent $\rightarrow$ so do <hab> and <(ah)b>

Given support threshold $\text{min\_sup} = 2$

<table>
<thead>
<tr>
<th>Seq-id</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>&lt;(bd)cb(ac)&gt;</td>
</tr>
<tr>
<td>20</td>
<td>&lt;(bf)(ce)b(fg)&gt;</td>
</tr>
<tr>
<td>30</td>
<td>&lt;(ah)(bf)abf&gt;</td>
</tr>
<tr>
<td>40</td>
<td>&lt;(be)(ce)d&gt;</td>
</tr>
<tr>
<td>50</td>
<td>&lt;a(bd)bcb(ade)&gt;</td>
</tr>
</tbody>
</table>
GSP

• GSP (Generalized Sequential Pattern) mining

• Outline of the method
  – Initially, every item in DB is a candidate of length-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 using Apriori
  – Repeat until no frequent sequence or no candidate can be found

• Major strength: Candidate pruning by Apriori
Finding Len-1 Seq Patterns

- Initial candidates
  - \(<a>\), \(<b>\), \(<c>\), \(<d>\), \(<e>\), \(<f>\), \(<g>\), \(<h>\)
- Scan database once
  - count support for candidates

\[
\text{min}_\text{sup} = 2
\]

<table>
<thead>
<tr>
<th>Seq-id</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>(&lt;(bd)cb(ac)&gt;)</td>
</tr>
<tr>
<td>20</td>
<td>(&lt;(bf)(ce)b(fg)&gt;)</td>
</tr>
<tr>
<td>30</td>
<td>(&lt;(ah)(bf)abf&gt;)</td>
</tr>
<tr>
<td>40</td>
<td>(&lt;(be)(ce)d&gt;)</td>
</tr>
<tr>
<td>50</td>
<td>(&lt;a(bd)bcb(ade)&gt;)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cand</th>
<th>Sup</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;a&gt;)</td>
<td>3</td>
</tr>
<tr>
<td>(&lt;b&gt;)</td>
<td>5</td>
</tr>
<tr>
<td>(&lt;c&gt;)</td>
<td>4</td>
</tr>
<tr>
<td>(&lt;d&gt;)</td>
<td>3</td>
</tr>
<tr>
<td>(&lt;e&gt;)</td>
<td>3</td>
</tr>
<tr>
<td>(&lt;f&gt;)</td>
<td>2</td>
</tr>
<tr>
<td>(&lt;g&gt;)</td>
<td>1</td>
</tr>
<tr>
<td>(&lt;h&gt;)</td>
<td>1</td>
</tr>
</tbody>
</table>
Generating Length-2 Candidates

51 length-2 Candidates

Without Apriori property, 8*8+8*7/2=92 candidates
Apriori prunes 44.57% candidates
Finding Len-2 Seq Patterns

• Scan database one more time, collect support count for each length-2 candidate
• There are 19 length-2 candidates which pass the minimum support threshold
  – They are length-2 sequential patterns
Generating Length-3 Candidates and Finding Length-3 Patterns

• Generate Length-3 Candidates
  – Self-join length-2 sequential patterns
    • <ab>, <aa> and <ba> are all length-2 sequential patterns → <aba> is a length-3 candidate
    • <(bd)>, <bb> and <db> are all length-2 sequential patterns → <(bd)b> is a length-3 candidate
  – 46 candidates are generated

• Find Length-3 Sequential Patterns
  – Scan database once more, collect support counts for candidates
  – 19 out of 46 candidates pass support threshold
The GSP Mining Process

5th scan: 1 cand. 1 length-5 seq. pat.

4th scan: 8 cand. 6 length-4 seq. pat.

3rd scan: 46 cand. 19 length-3 seq. pat. 20 cand. not in DB at all

2nd scan: 51 cand. 19 length-2 seq. pat. 10 cand. not in DB at all

1st scan: 8 cand. 6 length-1 seq. pat.

Cand. cannot pass sup. threshold

Cand. not in DB at all

\[ min\_sup = 2 \]

<table>
<thead>
<tr>
<th>Seq-id</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>&lt;(bd)cb(ac)&gt;</td>
</tr>
<tr>
<td>20</td>
<td>&lt;(bf)(ce)b(fg)&gt;</td>
</tr>
<tr>
<td>30</td>
<td>&lt;(ah)(bf)abf&gt;</td>
</tr>
<tr>
<td>40</td>
<td>&lt;(be)(ce)d&gt;</td>
</tr>
<tr>
<td>50</td>
<td>&lt;a(bd)bcb(ade)&gt;</td>
</tr>
</tbody>
</table>
The GSP Algorithm

- Take sequences in form of <x> as length-1 candidates
- Scan database once, find $F_1$, the set of length-1 sequential patterns
- Let $k=1$; while $F_k$ is not empty do
  - Form $C_{k+1}$, the set of length-$(k+1)$ candidates from $F_k$;
  - If $C_{k+1}$ is not empty, scan database once, find $F_{k+1}$, the set of length-$(k+1)$ sequential patterns
  - Let $k=k+1$;
Bottlenecks of GSP

• A huge set of candidates
  – 1,000 frequent length-1 sequences generate length-2 candidates!
  \[1000 \times 1000 + \frac{1000 \times 999}{2} = 1,499,500\]

• Multiple scans of database in mining

• Real challenge: mining long sequential patterns
  \[\sum_{i=1}^{100} \binom{100}{i} = 2^{100} - 1 \approx 10^{30}\]
  – An exponential number of short candidates
  – A length-100 sequential pattern needs \(10^{30}\) candidate sequences!
FreeSpan: Freq Pat-projected Sequential Pattern Mining

• The itemset of a seq pat must be frequent
  – Recursively project a sequence database into a set of smaller databases based on the current set of frequent patterns
  – Mine each projected database to find its patterns

<table>
<thead>
<tr>
<th>Sequence Database $SDB$</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; (bd) c b (ac) &gt;</td>
</tr>
<tr>
<td>&lt; (bf) (ce) b (fg) &gt;</td>
</tr>
<tr>
<td>&lt; (ah) (bf) a b f &gt;</td>
</tr>
<tr>
<td>&lt; (be) (ce) d &gt;</td>
</tr>
<tr>
<td>&lt; a (bd) b c b (ade) &gt;</td>
</tr>
</tbody>
</table>

All seq. pat. can be divided into 6 subsets:
• Seq. pat. containing item $f$
• Those containing $e$ but no $f$
• Those containing $d$ but no $e$ nor $f$
• Those containing $a$ but no $d$, $e$ or $f$
• Those containing $c$ but no $a$, $d$, $e$ or $f$
• Those containing only item $b$

$f_{list}$: b:5, c:4, a:3, d:3, e:3, f:2
From FreeSpan to PrefixSpan

• FreeSpan:
  – Projection-based: no candidate sequence needs to be generated
  – But, projection can be performed at any point in the sequence, and the projected sequences may not shrink much

• PrefixSpan
  – Projection-based
  – But only prefix-based projection: less projections and quickly shrinking sequences
Prefix and Suffix (Projection)

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

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Suffix (Prefix-Based Projection)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;a&gt;)</td>
<td>(&lt;(abc)(ac)d(cf))</td>
</tr>
<tr>
<td>(&lt;aa&gt;)</td>
<td>(&lt;(_bc)(ac)d(cf))</td>
</tr>
<tr>
<td>(&lt;ab&gt;)</td>
<td>(&lt;(_c)(ac)d(cf))</td>
</tr>
</tbody>
</table>
Mining Sequential Patterns by Prefix Projections

• Step 1: find length-1 sequential patterns
  – <a>, <b>, <c>, <d>, <e>, <f>

• Step 2: divide search space. The complete set of seq. pat. can be partitioned into 6 subsets:
  – The ones having prefix <a>;
  – The ones having prefix <b>;
  – ...
  – The ones having prefix <f>

<table>
<thead>
<tr>
<th>SID</th>
<th>sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>&lt;a(abc)(ac)d(cf)&gt;</td>
</tr>
<tr>
<td>20</td>
<td>&lt;(ad)c(bc)(ae)&gt;</td>
</tr>
<tr>
<td>30</td>
<td>&lt;(ef)(ab)(df)cb&gt;</td>
</tr>
<tr>
<td>40</td>
<td>&lt;eg(af)cbc&gt;</td>
</tr>
</tbody>
</table>
Finding Seq. Pat. with Prefix $<a>$

- Only need to consider projections w.r.t. $<a>$
  - $<a>$-projected database: $<(abc)(ac)d(cf)>, \ (<_d)c(bc)(ae)>, \ (<_b)(df)cb), \ (<_f)cbbc$.

- Find all the length-2 seq. pat. having prefix $<a>$: $<aa>, <ab>, <(ab)>, <ac>, <ad>, <af>$
  - Further partition into 6 subsets:
    - Having prefix $<aa>$;
    - ...
    - Having prefix $<af>$

<table>
<thead>
<tr>
<th>SID</th>
<th>sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>$&lt;a(abc)(ac)d(cf)&gt;$</td>
</tr>
<tr>
<td>20</td>
<td>$&lt;(ad)c(bc)(ae)&gt;$</td>
</tr>
<tr>
<td>30</td>
<td>$&lt;(ef)(ab)(df)cb&gt;$</td>
</tr>
<tr>
<td>40</td>
<td>$&lt;eg(af)cbbc&gt;$</td>
</tr>
</tbody>
</table>
Completeness of PrefixSpan

SDB

<table>
<thead>
<tr>
<th>SID</th>
<th>sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>&lt;(a(bc)(ac)d(cf)&gt;</td>
</tr>
<tr>
<td>20</td>
<td>&lt;(ad)c(bc)(ae)&gt;</td>
</tr>
<tr>
<td>30</td>
<td>&lt;(ef)(ab)(df)cb&gt;</td>
</tr>
<tr>
<td>40</td>
<td>&lt;eg(af)cb&gt;</td>
</tr>
</tbody>
</table>

Length-1 sequential patterns
<a>, <b>, <c>, <d>, <e>, <f>

Having prefix <a>

Having prefix <b>

Having prefix <c>, …, <f>

Length-2 sequential patterns
<aa>, <ab>, <(ab)>, <ac>, <ad>, <af>

<aa>-projected database
<(abc)(ac)d(cf)>
<(_d)c(bc)(ae)>
<(_b)(df)cb>
<(_f)cbc>

<af>-projected database

<aa>-proj. db
<af>-proj. db
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 bi-level projections
Constraints for Seq Pat Mining

• Item constraints
  – Find web log patterns only about online-bookstores

• Length constraints
  – Find patterns having at least 20 items

• Super pattern constraints
  – Find super patterns of “PC → digital camera”

• Aggregate constraints
  – Find patterns that the average price of items is over $100
More Constraints

• Regular expression constraints
  – Find patterns “starting from Yahoo homepage, search for hotels in Washington DC area”
  – Yahootravel(Washington DC|DC)(hotel|motel|lodging)
• Duration constraints
  – Find patterns about ±24 hours of a shooting
• Gap constraints
  – Find purchasing patterns such that “the gap between each consecutive purchases is less than 1 month”
Characterizations of Constraints

• Anti-monotonic constraints
  – If a sequence s satisfies $C \rightarrow$ so does every non-empty subsequence of s
  – Examples: $\text{sup}(s) \geq 5\%$, $\text{dur}(s) < 3$ months

• Monotonic constraints
  – If a sequence s satisfies $C \rightarrow$ so does every super sequence of s
  – Examples: $\text{len}(s) \geq 10$, super pattern constraints
## Characterization of Commonly Used Constraints

<table>
<thead>
<tr>
<th>Constraints</th>
<th>Characterizations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item constraint</td>
<td>Succinct; anti-monotonic or monotonic</td>
</tr>
<tr>
<td>Length constraints</td>
<td>Succinct; anti-monotonic or monotonic</td>
</tr>
<tr>
<td>Super-pattern constraints</td>
<td>Monotonic</td>
</tr>
<tr>
<td>Aggregate constraints</td>
<td>Some are neither anti-monotonic, monotonic</td>
</tr>
<tr>
<td>Regular expression constraints</td>
<td>Some are neither anti-monotonic, monotonic</td>
</tr>
<tr>
<td>Duration constraints</td>
<td>Anti-monotonic or monotonic</td>
</tr>
<tr>
<td>Gap constraints</td>
<td>Anti-monotonic</td>
</tr>
</tbody>
</table>
Prefix Monotone Property

• Prefix anti-monotonic constraint
  – A sequence \( s \) satisfies a constraint \( \rightarrow \) so does every non-empty prefix of \( s \)
  – An anti-monotonic constraint \( \rightarrow \) prefix anti-monotonic

• Prefix monotonic constraint
  – A sequence \( s \) satisfies a constraint \( \rightarrow \) so does every super sequence having \( s \) as a prefix
  – An monotonic constraint \( \rightarrow \) prefix monotonic
Regular Expression Constraints

• A sequence $s$ satisfies a regular expression constraint $\rightarrow$ every prefix of $s$ must be legal w.r.t. the corresponding deterministic finite automata

• Constraint “being legal” is prefix anti-monotonic
Pushing Prefix Constraints

- Only make projection for prefixes (potentially) satisfying the constraint
  - Constraint \(<a\{b, c\}^*d>\) \(\rightarrow\) Prune prefix \(<b>\)

- No constraint checking for patterns in the s-projected database if s satisfies the prefix monotonic constraint
  - Every pattern having a prefix \(<abc>\) satisfying the constraint \(<abc^*>\)

- For each projected sequence s in the P-projected database, \(<Ps>\) should contain a subsequence satisfying the constraint
  - Constraint \(<a\{b, c\}^*d>\) \(\rightarrow\) Don’t project any sequence not having d
Pushing Aggregate Constraints (1)

- Prune unpromising sequences
  - Constraint avg(s)>25
  - Prune a sequence if every item there has a value less than 25

- The technique can be applied recursively
  - In P-projected database, let x be the frequent item with the largest value. If avg(Px) <= 25, the P-projected database can be pruned
Pushing Aggregate Constraints (2)

• Prune unpromising patterns
  – Constraint \( \text{avg}(s) > 25 \)
  – A pattern \( P \) can be pruned if \( \text{avg}(P) < 25 \), and there is no pattern \( Q \) s.t. \( \text{avg}(Q) > 25 \) and \( \text{avg}(PQ) > 25 \)

• How to find \( Q \)?
  – The set of all frequent items in \( P \)-projected database that have values over 25
Graphs as Data Models

• Have been used extensively in many applications
  – Undirected graphs versus directed graphs
  – Weighted graphs, edge-labeled graphs, …
• Structured and unstructured models
  – Well structured in terms of nodes and edges
  – Random graphs
• Understanding graphs – How?
  – Micro view: nodes and edges
  – Macro view: global structures and properties
  – How to build the bridge?
WWW and Social Networks

- A huge (random) graph – millions or billions of nodes
- “Small world”
  - On average, sparse, small in-/out-degree per node
  - Short diameter in expectation
Biological Networks

- Example: Protein-protein interaction network
  - A few proteins interact with a large number of other proteins, while most proteins have only one or two links
- Thousands of nodes
- Somehow predictable structure
- A few networks exist

www.nd.edu/~networks/linked/newfile17.htm
Chemical Compound Structures

- Up to several thousand nodes
- Highly regular structures
- Millions of compounds
A Spectrum of Graph Databases

- Two major dimensions
  - Number of nodes per graph
  - Number of graphs in the database
- Another instance of power law?
  - Zipf distribution
- One dimension missing
  - The complexity of graphs!
Graph Mining

• Finding novel and potentially useful graph structures and properties from graph databases

• Graph structure mining
  – Frequent subgraph mining
  – Community mining

• Graph property mining
  – Changes of graph properties over time
Mining Frequent Tree/Graph Patterns

• Applications
  – XML documents can be modeled as trees
  – Chemical structures can be modeled as graphs
  – Frequent tree/graph patterns: common structural components

• The principle of frequent pattern mining works
  – Bottleneck: support counting – tree/graph isomorphism determination

• Representative methods
  – gSpan: a pattern-growth method
  – ADI-Mine: mining large disk-based graph databases
X-Market Customer Segmentation

- \{A, B, C, D, E\} is interesting – in each market, each customer is similar to at least three of the other four customers
  - Both the customers and the connectivity matter

\[G_f: \text{similarity graph in financial product market}\]

\[G_c: \text{similarity graph in consumer product market}\]
Weighted Similarity Graph?

- Cluster \{A, B, C, D, E\} and the connectivity information cannot be found from the weighted similarity graph!
Joint Mining in Bioinformatics

- Co-expressed genes from microarray data
- Interacting proteins from protein interaction data
- Connection: a protein is a product of a gene
- Joint mining of microarray data and protein interaction data
  - Both microarray data and protein interaction data are typically noisy
  - Can we validate clusters from mining the two types of data?
  - For many pathways, their genes exhibit a similar gene expression profile, and the protein products of the genes often interact
A Pattern from Yeast Data Sets
Intuition and Challenges

- Given a set of vertices and multiple graphs on the vertices, find the maximal subsets of vertices whose induced subgraphs in each graph are almost complete.
- A weighted graph approach cannot capture the same clusters and information.
Quasi-Complete Graphs/Cliques

- A connected graph is a $\gamma$-complete graph ($0 < \gamma \leq 1$) if every vertex in the graph has a degree at least $\gamma(|V(G)|-1)$
  - A 1-complete graph is a conventional complete graph
- In a graph $G$, a subset of vertices $S \subseteq V(G)$ is a $\gamma$-quasi-clique if $G(S)$ is a $\gamma$-complete graph, and no proper superset of $S$ has this property
Quasi-Complete Graphs Monotonic?

- If $G(S)$ is a complete graph, then for any non-empty $S' \subseteq S$, $G(S')$ is also a complete graph.
- The monotonicity does not hold for $\gamma$-complete graph if $\gamma < 1$.

$\gamma = 0.8$
Realizable Upper Bound on Diameter

\[ \text{diam}(G) \begin{cases} = 1 \\ \leq 2 \\ \leq 3 \left\lfloor \frac{n}{\gamma(n-1)+1} \right\rfloor - 3 \\ \leq 3 \left\lfloor \frac{n}{\gamma(n-1)+1} \right\rfloor - 2 \\ \leq 3 \left\lfloor \frac{n}{\gamma(n-1)+1} \right\rfloor - 1 \\ \leq n - 1 \end{cases} \]

if \( 1 \geq \gamma > \frac{n-2}{n-1} \)

if \( \frac{n-2}{n-1} \geq \gamma \geq \frac{1}{2} \)

if \( \frac{1}{2} > \gamma \geq \frac{2}{n-1} \) and

\( n \mod (\gamma(n-1) + 1) = 0 \)

if \( \frac{1}{2} > \gamma \geq \frac{2}{n-1} \) and

\( n \mod (\gamma(n-1) + 1) = 1 \)

if \( \frac{1}{2} > \gamma \geq \frac{2}{n-1} \) and

\( n \mod (\gamma(n-1) + 1) \geq 2 \)

if \( \gamma = \frac{1}{n-1} \)
How to Tune Parameter $\gamma$?

- Good News: the diameter of a $\gamma$-complete graph is relatively insensitive to $\gamma$
Cross-Graph Quasi-Cliques

• A set of graphs $G_1, \ldots, G_n$ and parameters $\gamma_1, \ldots, \gamma_n$ such that $V(G_1) = \ldots = V(G_n) = U$ and $0 < \gamma_1, \ldots, \gamma_n \leq 1$

• A subset $S \subseteq U$ is a cross-graph quasi-clique (CGQC) if
  – (Quasi-complete) $G_i(S)$ is a $\gamma_i$-complete graph for $1 \leq i \leq n$
  – (Maximal) No any proper superset of $S$ has the property
  – (Significant) $S$ has at least $\min_S$ vertices
CGQC and Quasi-Cliques

• If n=1 (only one member graph), a CGQC is a quasi-clique

• Generally, a CGQC may not be a quasi-clique in member graphs – may not be locally maximal

$\gamma_1 = \gamma_2 = 0.5$, \{a, b, d\} is a CGQC
How to Mine CGQCs?

- Generally, we cannot mine from an integrated graph
  - The integrated graph method works if $\gamma_1 = \ldots = \gamma_n = 1$
  - Generally, we have to take a joint-mining approach

- The problem of counting the number of cross-graph quasi-cliques is in #P-Complete

- A difficult problem!
Algorithm Crochet

• Depth-first enumeration of vertex subsets
  – Guarantee the completeness of answers
• At each step, be efficient!
  – Aggressively reduce graphs
  – Dynamically choose the order to search children
  – Sharply prune futile subtrees
Depth-First Set Enumeration

- Use the popularly used set enumeration tree
Reducing Vertices and Edges

• Reducing vertices
  – For vertex v, if deg(v) in one member graph is insufficient to form a quasi-complete subgraph, then v can be pruned
  – The pruning can be applied repeatedly since removing a vertex may reduce the degrees of some other vertices

• Reducing edges
  – If there exists a member graph G_i with \( \gamma_i = 1 \), then for any vertices u and v that are not connected in G_i, (u, v) can be removed from the other member graphs

• Reducing vertices and edges iteratively
Combining Graphs

• If $\gamma_1=\gamma_2=1$, $S$ is a cross-graph quasi-clique if and only if $G_1(S)$ and $G_2(S)$ are both complete graphs
  – $S$ must be a clique in $G=(E_1 \cap E_2, V_1 \cap V_2)$
• All the member graphs with $\gamma_i=1$ can be combined into one graph
• Reducing graphs can make the mining faster
Pruning Set Enumeration Trees

• At node $S$, only the vertices that are “well connected” to $S$ should be used to expand $S$
  – Using the diameter of quasi-cliques as bounds, details in the paper

• At node $S$, let $S'$ be the set of all vertices that are “well connected” to $S$, if $S \cup S'$ does not contain any quasi-clique, then the subtree of $S$ can be pruned

• Search the children nodes in the well-connected-ness descending order
Sequences and Partial Orders

Sequential patterns:
- CHK → MMK → MORT → RESP
- CHK → MMK → MORT → BROK
- CHK → RRSP → MORT → RESP
- CHK → RRSP → MORT → BROK

<table>
<thead>
<tr>
<th>Cid</th>
<th>Sequence of account opening</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>CHK → MMK → RRSP → MORT → RESP → BROK</td>
</tr>
<tr>
<td>2</td>
<td>CHK → RRSP → MMK → MORT → RESP → BROK</td>
</tr>
<tr>
<td>3</td>
<td>MMK → CHK → BROK → RESP → RRSP</td>
</tr>
<tr>
<td>4</td>
<td>CHK → MMK → RRSP → MORT → BROK → RESP</td>
</tr>
</tbody>
</table>

Account type:
- CHK: Checking account
- MMK: Money market
- RRSP: Retirement Savings Plan
- MORT: Mortgage
- RESP: Registered Education Savings Plan
- BROK: Brokerage
Why Frequent Orders?

- Frequent orders capture more thorough information than sequential patterns
- Many important applications
  - Bioinformatics: order-preserving clustering of microarray data
  - Web mining and market basket analysis: modeling customer purchase behaviors
  - Network management and intrusion detection: frequent routing paths, signatures for intrusions
  - Preference-based services: partial orders from ranking data
Why Mining Orders Difficult?

- Use sequential patterns to assemble frequent partial orders?
  - One frequent closed partial order may summarize a few sequential patterns
  - Assembling can be costly

Sequential patterns:
- CHK → MMK → MORT → RESP
- CHK → MMK → MORT → BROK
- CHK → RRSP → MORT → RESP
- CHK → RRSP → MORT → BROK

Diagram:
- Checking account (CHK)
  - Money market account (MMK)
    - Mortgage account (MORT)
      - Brokerage account (BROK)
      - Registered Education Savings Plan (RESP)
  - Retirement Savings Plan (RRSP)
Model

- A sequence $s$ induces a full order $R_1$, if $R_1 \rightarrow R_2$, where $R_2$ is a partial order, then $R_1$ is said to support $R_2$.
- The support of a partial order $R$ in a sequence database is the number of sequences supporting $R$ in the database.
- An order $R$ is closed if there exists no any $R' \rightarrow R$ and $\text{sup}(R) = \text{sup}(R')$.
- Given a minimum support threshold, order $R$ is a frequent closed partial order if it is closed and passes the support threshold.
Ideas

• Depth-first search to generate frequent closed partial orders in transitive reduction
  – Transitive reduction is a succinct representation of partial orders
• Pruning infrequent items, edges and partial orders
• Pruning forbidden edges
• Extracting transitive reductions of frequent partial orders directly
Interesting Orders

(b) A pattern in data set BreastCancer (support=224)

(a) A pattern in data set Yeast (support=80)

(e) Another pattern in data set Snake (support=80).
Freq pat Mining: Achievements

• An important task in data mining
• Frequent pattern mining methodology
  – Candidate generation & test vs. frequent-pattern growth
  – Vertical vs. horizontal format
  – Various optimization methods: database partition, scan reduction, hash tree, sampling, border computation, clustering, etc.
Search Strategies

• Breadth-first vs. depth-first search
  – Apriori-based vs. pattern growth methods
  – Pattern-growth methods have good performance in large and dense databases

• How to search?
  – Reduce recursion/counting cost
  – Compress database
Mining Various Patterns

- Mining closed frequent itemsets and max-patterns
- Mining multi-level, multi-dimensional frequent patterns with flexible support constraints
- Constraint pushing for mining optimization
- From frequent patterns to sequential patterns, correlation and causality
Applications

- Association-based classification
- Iceberg cube computation
- Mining sequential patterns
- Mining partial periodicity, cyclic associations, etc.
- Mining frequent structures, trends, etc.
What Is the Next Step?

• New applications of frequent pattern mining
  – Extensions
  – Use frequent pattern mining ideas to solve other problems – great!

• New types of frequent patterns
Resources for FPM Research

• Data sets:
  – UCI data sets and some real data sets
  – Archived at FIMI workshop website

• Implementations
  – Some source codes are available on the web, but unnecessarily are good implementations
Reading List (1)

- J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. SIGMOD'00.
- R. J. Bayardo. Efficiently mining long patterns from databases. SIGMOD'98.
- J. Pei, J. Han, and R. Mao. CLOSET: an efficient algorithm for mining frequent closed itemsets. In DMKD'00, Dallas, USA, May 2000.
- R. Ng, L. V. S. Lakshmanan, J. Han and A. Pang, Exploratory Mining and Pruning Optimizations of Constrained Associations Rules, SIGMOD’98.
- J. Pei, J. Han, and L. V. S. Lakshmanan, Mining Frequent Itemsets withConvertible Constraints, ICDE'01.
- J. Han and J. Pei, Mining Frequent Patterns by Pattern-Growth: Methodology and Implications, ACM SIGKDD Explorations (Special Issue on Scalable Data Mining Algorithms), December 2000.
Reading List (2)

- J. Pei and J. Han, Constrained Frequent Pattern Mining: A Pattern-Growth View, ACM SIGKDD Explorations (Special Issue on Constraints in Data Mining), June 2002.
- J. Pei, et al., PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth. ICDE'01
- X. Yan and J. Han. gSpan: Graph-Based Substructure Pattern Mining. ICDM 2002.