Mining Association Rules in Large Databases

- Association rule mining
- Algorithms for scalable mining of (single-dimensional Boolean) association rules in transactional databases
- Mining various kinds of association/correlation rules
- Constraint-based association mining
- Sequential pattern mining
- Applications/extensions of frequent pattern mining
- Summary
What Is Association Mining?

• Association rule mining:
  – Finding frequent patterns, associations, correlations, or causal structures among sets of items or objects in transaction databases, relational databases, and other information repositories.
  – Frequent pattern: pattern (set of items, sequence, etc.) that occurs frequently in a database [AIS93]

• Motivation: finding regularities in data
  – What products were often purchased together? — Beer and diapers?!
  – What are the subsequent purchases after buying a PC?
  – What kinds of DNA are sensitive to this new drug?
  – Can we automatically classify web documents?

Why Is Association Mining Important?

• Foundation for many essential data mining tasks
  – Association, correlation, causality
  – Sequential patterns, temporal or cyclic association, partial periodicity, spatial and multimedia association
  – Associative classification, cluster analysis, iceberg cube, fascicles (semantic data compression)

• Broad applications
  – Basket data analysis, cross-marketing, catalog design, sale campaign analysis
  – Web log (click stream) analysis, DNA sequence analysis, etc.
Basic Concepts: Association Rules

- Itemset $X = \{x_1, \ldots, x_k\}$
- Find all the rules $X \Rightarrow Y$ with min confidence and support
  - $\text{support}$, $s$, probability that a transaction contains $X \cup Y$
  - $\text{confidence}$, $c$, conditional probability that a transaction having $X$ also contains $Y$.

Let $\text{min\_support} = 50\%$, $\text{min\_conf} = 50\%$:
$A \Rightarrow C$ (50\%, 66.7\%)
$C \Rightarrow A$ (50\%, 100\%)

Mining Association Rules: Example

<table>
<thead>
<tr>
<th>Transaction-id</th>
<th>Items bought</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>A, B, C</td>
</tr>
<tr>
<td>20</td>
<td>A, C</td>
</tr>
<tr>
<td>30</td>
<td>A, D</td>
</tr>
<tr>
<td>40</td>
<td>B, E, F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Frequent pattern</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A)</td>
<td>75%</td>
</tr>
<tr>
<td>(B)</td>
<td>50%</td>
</tr>
<tr>
<td>(C)</td>
<td>50%</td>
</tr>
<tr>
<td>(A, C)</td>
<td>50%</td>
</tr>
</tbody>
</table>

For rule $A \Rightarrow C$:
$\text{support} = \text{support}((\{A\} \cup \{C\}) = 50\%$
$\text{confidence} = \text{support}((\{A\} \cup \{C\})/\text{support}(\{A\}) = 66.6\%$
Mining Association Rules: What We Need to Know

• Goal: Rules with high support/confidence
• How to compute?
  – Support: Find sets of items that occur frequently
  – Confidence: Find frequency of subsets of supported itemsets
• If we have all frequently occurring sets of items (frequent itemsets), we can compute support and confidence!

Mining Association Rules in Large Databases

• Association rule mining
• Algorithms for scalable mining of (single-dimensional Boolean) association rules in transactional databases
• Mining various kinds of association/correlation rules
• Constraint-based association mining
• Sequential pattern mining
• Applications/extensions of frequent pattern mining
• Summary
Apriori: A Candidate Generation-and-Test Approach

- **Any subset of a frequent itemset must be frequent**
  - if \{beer, diaper, nuts\} is frequent, so is \{beer, diaper\}
  - Every transaction having \{beer, diaper, nuts\} also contains \{beer, diaper\}
- **Apriori pruning principle**: If there is any itemset which is infrequent, its superset should not be generated/tested!
- Method:
  - generate length \(k+1\) candidate itemsets from length \(k\) frequent itemsets, and
  - test the candidates against DB
- Performance studies show its efficiency and scalability

The Apriori Algorithm—An Example

Database TDB

<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>

<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>

\(L_2\)

<table>
<thead>
<tr>
<th>Itemset</th>
<th>sup</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A, C)</td>
<td>2</td>
</tr>
<tr>
<td>(B, C)</td>
<td>2</td>
</tr>
<tr>
<td>(B, E)</td>
<td>3</td>
</tr>
<tr>
<td>(C, E)</td>
<td>2</td>
</tr>
</tbody>
</table>

\(L_3\)

<table>
<thead>
<tr>
<th>Itemset</th>
<th>sup</th>
</tr>
</thead>
<tbody>
<tr>
<td>(B, C, E)</td>
<td>2</td>
</tr>
</tbody>
</table>

1st scan: \(C_1\)

2nd scan: \(C_2\)

3rd scan: \(C_3\)

4th scan: \(C_4\)

5th scan: \(C_5\)

6th scan: \(C_6\)

7th scan: \(C_7\)

8th scan: \(C_8\)

9th scan: \(C_9\)

10th scan: \(C_{10}\)
The Apriori Algorithm

- **Pseudo-code:**
  
  $C_k$: Candidate itemset of size $k$
  $L_k$: frequent itemset of size $k$

  $$L_f = \{\text{frequent items}\};$$
  
  for (k = 1; $L_k \neq \emptyset$; k++) do begin
  
  $C_{k+1}$ = 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
  
  end

  return $\bigcup_k L_k$;

Important Details of Apriori

- How to generate candidates?
  - Step 1: self-joining $L_k$
  - Step 2: pruning

- How to count supports of candidates?

- Example of Candidate-generation
  - $L_3=\{abc, abd, acd, ace, bcd\}$
  - Self-joining: $L_3 \ast L_3$
    - $abcd$ from $abc$ and $abd$
    - $acde$ from $acd$ and $ace$
  - 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-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 $p.item_1=q.item_1, ..., p.item_{k-2}=q.item_{k-2}, p.item_{k-1} < q.item_{k-1}$**
- 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 $C_k$**

---

How to Count Supports of Candidates?

- 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**
  - **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
Efficient Implementation of Apriori in SQL

- Hard to get good performance out of pure SQL (SQL-92) based approaches alone

- Make use of object-relational extensions like UDFs, BLOBs, Table functions etc.
  - Get orders of magnitude improvement

Challenges of Frequent Pattern Mining

- Challenges
  - Multiple scans of transaction database
  - Huge number of candidates
  - Tedious workload of support counting for candidates
- Improving Apriori: general ideas
  - Reduce passes of transaction database scans
  - Shrink number of candidates
  - Facilitate support counting of candidates

DIC: Reduce Number of Scans

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

Partition: Scan Database Only Twice

- Any itemset that is potentially frequent in DB must be frequent in at least one of the partitions of DB
  - Scan 1: partition database and find local frequent patterns
  - Scan 2: consolidate global frequent patterns

Sampling for Frequent Patterns

- Select a sample of original database, mine frequent patterns within sample using Apriori
- Scan database once to verify frequent itemsets found in 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. Sampling large databases for association rules. In *VLDB’96*
DHP: Reduce the Number of Candidates

- A k-itemset whose corresponding hashing bucket count is below the threshold cannot be frequent
  - Candidates: a, b, c, d, e
  - Hash entries: {ab, ad, ae} {bd, be, de} ...
  - Frequent 1-itemset: a, b, d, e
  - ab is not a candidate 2-itemset if the sum of count of {ab, ad, ae} is below support threshold

Eclat/MaxEclat and VIPER: Exploring Vertical Data Format

- Use tid-list, the list of transaction-ids containing an itemset
- Compression of tid-lists
  - Itemset A: t1, t2, t3, sup(A)=3
  - Itemset B: t2, t3, t4, sup(B)=3
  - Itemset AB: t2, t3, sup(AB)=2
- Major operation: intersection of tid-lists
- M. Zaki et al. New algorithms for fast discovery of association rules. In KDD’97
- P. Shenoy et al. Turbo-charging vertical mining of large databases. In SIGMOD’00
Bottleneck of Frequent-pattern Mining

- Multiple database scans are **costly**
- Mining long patterns needs many passes of scanning and generates lots of candidates
  - To find frequent itemset $i_1i_2...i_{100}$
    - # of scans: 100
    - # of Candidates: $(100^1) + (100^2) + ... + (10^0,0) = 2^{100} - 1 = 1.27\times10^{30}$

- Bottleneck: candidate-generation-and-test
- Can we avoid candidate generation?
Mining Frequent Patterns Without Candidate Generation

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

Construct FP-tree from a Transaction Database

<table>
<thead>
<tr>
<th>TID</th>
<th>Items bought</th>
<th>(ordered) frequent items</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>{f, a, c, d, g, i, m, p}</td>
<td>{f, c, a, m, p}</td>
</tr>
<tr>
<td>200</td>
<td>{a, b, c, f, l, m, o}</td>
<td>{f, c, a, b, m}</td>
</tr>
<tr>
<td>300</td>
<td>{b, f, h, j, o, w}</td>
<td>{f, b}</td>
</tr>
<tr>
<td>400</td>
<td>{b, c, k, s, p}</td>
<td>{c, b, p}</td>
</tr>
<tr>
<td>500</td>
<td>{a, f, c, e, l, p, m, n}</td>
<td>{f, c, a, m, p}</td>
</tr>
</tbody>
</table>

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

Header Table

<table>
<thead>
<tr>
<th>Item</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>4</td>
</tr>
<tr>
<td>c</td>
<td>4</td>
</tr>
<tr>
<td>a</td>
<td>3</td>
</tr>
<tr>
<td>b</td>
<td>3</td>
</tr>
<tr>
<td>m</td>
<td>3</td>
</tr>
<tr>
<td>p</td>
<td>3</td>
</tr>
</tbody>
</table>

F-list = f-c-a-b-m-p

<table>
<thead>
<tr>
<th>Item</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>c:3</td>
<td></td>
</tr>
<tr>
<td>b:1</td>
<td></td>
</tr>
<tr>
<td>a:3</td>
<td></td>
</tr>
<tr>
<td>m:2</td>
<td></td>
</tr>
<tr>
<td>p:2</td>
<td></td>
</tr>
<tr>
<td>f:4</td>
<td></td>
</tr>
<tr>
<td>c:1</td>
<td></td>
</tr>
<tr>
<td>b:1</td>
<td></td>
</tr>
<tr>
<td>b:1</td>
<td></td>
</tr>
<tr>
<td>p:1</td>
<td></td>
</tr>
<tr>
<td>m:1</td>
<td></td>
</tr>
</tbody>
</table>

min_support = 3
Benefits of the FP-tree Structure

- Completeness
  - Preserve complete information for frequent pattern mining
  - Never break a long pattern of any transaction
- Compactness
  - Reduce irrelevant info—infrequent items are gone
  - Items in frequency descending order: the more frequently occurring, the more likely to be shared
  - Never be larger than the original database (not count node-links and the count field)
  - For Connect-4 DB, compression ratio could be over 100

Partition Patterns and Databases

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

- For each pattern-base
  - Accumulate the count for each item in the base
  - Construct the FP-tree for the frequent items of the pattern base
Recursion: Mining Each Conditional FP-tree

A Special Case: Single Prefix Path in FP-tree

- Suppose a (conditional) FP-tree $T$ has a shared single prefix-path $P$
- Mining can be decomposed into two parts
  - Reduction of the single prefix path into one node
  - Concatenation of the mining results of the two parts
Mining Frequent Patterns With FP-trees

- Idea: Frequent pattern growth
  - Recursively grow frequent patterns by pattern and database partition
- Method
  - For each frequent item, construct its conditional pattern-base, and then its conditional FP-tree
  - Repeat the process on each newly created conditional FP-tree
  - Until the resulting FP-tree is empty, or it contains only one path—single path will generate all the combinations of its sub-paths, each of which is a frequent pattern

Scaling FP-growth by DB Projection

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

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

FP-Growth vs. Apriori: Scalability

With the Support Threshold

Data set T25I20D10K
FP-Growth vs. Tree-Projection: Scalability with the 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 ops—counting local freq items and building sub FP-tree, no pattern search and matching
Implications of the Methodology

- Mining closed frequent itemsets and max-patterns
  - CLOSET (DMKD’00)
- Mining sequential patterns
  - FreeSpan (KDD’00), PrefixSpan (ICDE’01)
- Constraint-based mining of frequent patterns
  - Convertible constraints (KDD’00, ICDE’01)
- Computing iceberg data cubes with complex measures
  - H-tree and H-cubing algorithm (SIGMOD’01)

Max-patterns

- Frequent pattern \{a_1, \ldots, a_{100}\} \Rightarrow (^{100}_1) + (^{100}_2) + \ldots + (^{100}_0) = 2^{100} - 1 = 1.27 \times 10^{30}
  frequent sub-patterns!
- Max-pattern: frequent patterns without
  proper frequent super pattern
  - BCDE, ACD are max-patterns
  - BCD is not a max-pattern

<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
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

- Potential max-patterns
- Since BCDE is a max-pattern, no need to check BCD, BDE, CDE in later scan
- R. Bayardo. Efficiently mining long patterns from databases. In SIGMOD'98

Frequent Closed Patterns

- Conf(ac→d)=100%  ➔ record acd only
- For frequent itemset X, if there exists no item y s.t. every transaction containing X also contains y, then X is a frequent closed pattern
  - “acd” is a frequent closed pattern
- Concise rep. of freq pats
- Reduce # of patterns and rules
- N. Pasquier et al. In ICDT’99
Mining Frequent Closed Patterns: CLOSET

- **Flist**: list of all frequent items in support ascending 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
- J. Pei, J. Han & R. Mao. CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets”, DMKD’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>

Mining Frequent Closed Patterns: CHARM

- **Use vertical data format**: t(AB)={(T₁, T₁₂, ...)}
- **Derive closed pattern based on vertical intersections**
  - t(X)=t(Y): X and Y always happen together
  - t(X)⊂t(Y): transaction having X always has Y
- **Use difset to accelerate mining**
  - Only keep track of difference of tids
  - t(X)=(T₁, T₂, T₃), t(XY)=(T₁, T₃)
  - Diffset(XY, X)=(T₃)
- M. Zaki. CHARM: An Efficient Algorithm for Closed Association Rule Mining, CS-TR99-10, Rensselaer Polytechnic Institute
- M. Zaki, Fast Vertical Mining Using Diffssets, TR01-1, Department of Computer Science, Rensselaer Polytechnic Institute
Visualization of Association Rules: Pane Graph

Visualization of Association Rules: Rule Graph
Mining Association Rules in Large Databases

- Association rule mining
- Algorithms for scalable mining of (single-dimensional Boolean) association rules in transactional databases
- Mining various kinds of association/correlation rules
- Constraint-based association mining
- Sequential pattern mining
- Applications/extensions of frequent pattern mining
- Summary

Mining Various Kinds of Rules or Regularities

- Multi-level, quantitative association rules, correlation and causality, ratio rules, sequential patterns, emerging patterns, temporal associations, partial periodicity
- Classification, clustering, iceberg cubes, etc.
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

<table>
<thead>
<tr>
<th>uniform support</th>
<th>reduced support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1</td>
<td></td>
</tr>
<tr>
<td>min_sup = 5%</td>
<td>Level 1</td>
</tr>
<tr>
<td>Milk</td>
<td></td>
</tr>
<tr>
<td>[support = 10%]</td>
<td>min_sup = 5%</td>
</tr>
<tr>
<td>Level 2</td>
<td></td>
</tr>
<tr>
<td>min_sup = 5%</td>
<td>Level 2</td>
</tr>
<tr>
<td>2% Milk</td>
<td></td>
</tr>
<tr>
<td>[support = 6%]</td>
<td>min_sup = 3%</td>
</tr>
<tr>
<td>Skin Milk</td>
<td></td>
</tr>
<tr>
<td>[support = 4%]</td>
<td></td>
</tr>
</tbody>
</table>

ML/MD Associations with 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 may be specified individually and have higher priority

CS590D
Multi-dimensional Association

- Single-dimensional rules:
  \[\text{buys}(X, \text{“milk”}) \Rightarrow \text{buys}(X, \text{“bread”})\]
- Multi-dimensional rules: \(\geq 2\) dimensions or predicates
  - Inter-dimension assoc. rules \(\text{(no repeated predicates)}\)
    \[\text{age}(X, \text{“19-25”}) \land \text{occupation}(X, \text{“student”}) \Rightarrow \text{buys}(X, \text{“coke”})\]
  - Hybrid-dimension assoc. rules \(\text{(repeated predicates)}\)
    \[\text{age}(X, \text{“19-25”}) \land \text{buys}(X, \text{“popcorn”}) \Rightarrow \text{buys}(X, \text{“coke”})\]
- Categorical Attributes
  - finite number of possible values, no ordering among values
- Quantitative Attributes
  - numeric, implicit ordering among values

Multi-level Association: Redundancy Filtering

- Some rules may be redundant due to “ancestor” relationships between items.
- Example
  - milk \(\Rightarrow\) wheat bread \[\text{[support = 8\%, confidence = 70\%]}\]
  - 2\% milk \(\Rightarrow\) wheat bread \[\text{[support = 2\%, confidence = 72\%]}\]
- We say the first rule is an ancestor of the second rule.
- A rule is redundant if its support is close to the “expected” value, based on the rule’s ancestor.
Multi-Level Mining: Progressive Deepening

- A top-down, progressive deepening approach:
  - First mine high-level frequent items:
    - milk (15%), bread (10%)
  - Then mine their lower-level “weaker” frequent itemsets:
    - 2% milk (5%), wheat bread (4%)
- Different min_support threshold across multi-levels lead to different algorithms:
  - If adopting the same min_support across multi-levels
    then toss t if any of t's ancestors is infrequent.
  - If adopting reduced min_support at lower levels
    then examine only those descendents whose ancestor’s support is frequent/non-negligible.

Techniques for Mining MD Associations

- Search for frequent k-predicate set:
  - Example: \{age, occupation, buys\} is a 3-predicate set
  - Techniques can be categorized by how age are treated
1. Using static discretization of quantitative attributes
   - Quantitative attributes are statically discretized by using predefined concept hierarchies
2. Quantitative association rules
   - Quantitative attributes are dynamically discretized into “bins” based on the distribution of the data
3. Distance-based association rules
   - This is a dynamic discretization process that considers the distance between data points
Static Discretization of Quantitative Attributes

- Discretized prior to mining using concept hierarchy.
- Numeric values are replaced by ranges.
- In relational database, finding all frequent k-predicate sets will require k or k+1 table scans.
- Data cube is well suited for mining.
- The cells of an n-dimensional cuboid correspond to the predicate sets.
- Mining from data cubes can be much faster.

Quantitative Association Rules

- Numeric attributes are *dynamically* discretized
  - Such that the confidence or compactness of the rules mined is maximized
- 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
- Example
  \[
  \text{age}(X,"30-34") \land \text{income}(X,"24K - 48K") \\
  \Rightarrow \text{buys}(X,"high resolution TV")
  \]
# Mining Distance-based Association Rules

- Binning methods do not capture the semantics of interval data

<table>
<thead>
<tr>
<th>Price($)</th>
<th>Equi-width (width $10)</th>
<th>Equi-depth (depth 2)</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>[50,53]</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></td>
</tr>
<tr>
<td>53</td>
<td>[51,60]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Distance-based partitioning, more meaningful discretization considering:
  - density/number of points in an interval
  - “closeness” of points in an interval

# Interestingness Measure: Correlations (Lift)

- *play basketball* ⇒ *eat cereal* [40%, 66.7%] is misleading
  - The overall percentage of students eating cereal is 75% which is higher than 66.7%.

- *play basketball* ⇒ *not eat cereal* [20%, 33.3%] is more accurate, although with lower support and confidence

- Measure of dependent/correlated events: lift

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

<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>
Mining Association Rules in Large Databases

- Association rule mining
- Algorithms for scalable mining of (single-dimensional Boolean) association rules in transactional databases
- Mining various kinds of association/correlation rules
- Constraint-based association mining
- Sequential pattern mining
- Applications/extensions of frequent pattern mining
- Summary

Constraint-based Data Mining

- Finding all the patterns in a database autonomously? — unrealistic!
  - The patterns could be too many but not focused!
- Data mining should be an interactive process
  - User directs what to be mined using a data mining query language (or a graphical user interface)
- Constraint-based mining
  - User flexibility: provides constraints on what to be mined
  - System optimization: explores such constraints for efficient mining—constraint-based 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 Vancouver in Dec.'00
- **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: min_support ≥ 3%, min_confidence ≥ 60%

Constrained Mining vs. Constraint-Based Search

- Constrained mining vs. constraint-based search/reasoning
  - Both are aimed at reducing search space
  - Finding all patterns satisfying constraints vs. finding some (or one) answer in constraint-based search in AI
  - Constraint-pushing vs. heuristic search
  - It is an interesting research problem on how to integrate them
- Constrained mining vs. query processing in DBMS
  - Database query processing requires to find all
  - Constrained pattern mining shares a similar philosophy as pushing selections deeply in query processing
Constrained Frequent Pattern Mining: A Mining Query Optimization Problem

• Given a frequent pattern mining query with a set of constraints $C$, the algorithm should be
  – sound: it only finds frequent sets that satisfy the given constraints $C$
  – complete: all frequent sets satisfying the given constraints $C$ are found

• A naïve solution
  – First find all frequent sets, and then test them for constraint satisfaction

• More efficient approaches:
  – Analyze the properties of constraints comprehensively
  – Push them as deeply as possible inside the frequent pattern computation.
Anti-Monotonicity in Constraint-Based Mining

- Anti-monotonicity
  - When an itemset $S$ violates the constraint, so does any of its superset
  - $\text{sum}(S.\text{Price}) \leq v$ is anti-monotone
  - $\text{sum}(S.\text{Price}) \geq v$ is not anti-monotone
- Example. C: range(S,profit) $\leq 15$ is anti-monotone
  - 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>

Which Constraints Are Anti-Monotone?

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Antimonotone</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>$\text{min}(S) \leq v$</td>
<td>no</td>
</tr>
<tr>
<td>$\text{min}(S) \geq v$</td>
<td>yes</td>
</tr>
<tr>
<td>$\text{max}(S) \leq v$</td>
<td>yes</td>
</tr>
<tr>
<td>$\text{max}(S) \geq v$</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>$\text{sum}(S) \leq v (a \in S, a \geq 0)$</td>
<td>yes</td>
</tr>
<tr>
<td>$\text{sum}(S) \geq v (a \in S, a \geq 0)$</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>$\text{avg}(S) \oplus V, \oplus \in {+, -, }$</td>
<td>convertible</td>
</tr>
<tr>
<td>support($S$) $\geq \xi$</td>
<td>yes</td>
</tr>
<tr>
<td>support($S$) $\leq \xi$</td>
<td>no</td>
</tr>
</tbody>
</table>
Monotonicity in Constraint-Based Mining

- Monotonicity
  - When an itemset \( S \) satisfies the constraint, so does any of its superset
    - \( \text{sum}(S.\text{Price}) \geq v \) is monotone
    - \( \text{min}(S.\text{Price}) \leq v \) is monotone

- Example. C: \( \text{range}(S.\text{profit}) \geq 15 \)
  - Itemset \( ab \) satisfies 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>

### Which Constraints Are Monotone?

<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>( \text{min}(S) \leq v )</td>
<td>yes</td>
</tr>
<tr>
<td>( \text{min}(S) \geq v )</td>
<td>no</td>
</tr>
<tr>
<td>( \text{max}(S) \leq v )</td>
<td>no</td>
</tr>
<tr>
<td>( \text{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>( \text{avg}(S) \leq v \ (0 \leq 0 \leq \infty, 0 \leq \infty) )</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:
  - Given $A_r$, the set of items satisfying a succinctness constraint $C$, then any set $S$ satisfying $C$ is based on $A_r$, i.e., $S$ contains a subset belonging to $A_r$
  - Idea: 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

Which Constraints Are Succinct?

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Succinct</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v \in S$</td>
<td>yes</td>
</tr>
<tr>
<td>$S \sqsupseteq V$</td>
<td>yes</td>
</tr>
<tr>
<td>$S \sqsubseteq V$</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>$\sum(S) \leq v(a \in S, a \geq 0)$</td>
<td>no</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>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>
The Apriori Algorithm — Example

Naïve Algorithm: Apriori + Constraint
The Constrained Apriori Algorithm: Push an Anti-monotone Constraint Deep

Database D

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1 3 4</td>
</tr>
<tr>
<td>200</td>
<td>2 3 5</td>
</tr>
<tr>
<td>300</td>
<td>1 2 3 5</td>
</tr>
<tr>
<td>400</td>
<td>2 5</td>
</tr>
</tbody>
</table>

Scan D

C₁

itemset sup.

\(\{1\}\) 2
\(\{2\}\) 3
\(\{3\}\) 3
\(\{4\}\) 1
\(\{5\}\) 3

L₁

itemset sup.

\(\{1\}\) 2
\(\{2\}\) 3
\(\{3\}\) 3
\(\{5\}\) 3

C₂

itemset sup.

\(\{1 2\}\) 1
\(\{1 3\}\) 2
\(\{1 5\}\) 1
\(\{2 3\}\) 2
\(\{2 5\}\) 3
\(\{3 5\}\) 2

Scan D

L₂

itemset sup.

\(\{1 2\}\) 1
\(\{1 3\}\) 2
\(\{1 5\}\) 1
\(\{2 3\}\) 2
\(\{2 5\}\) 3
\(\{3 5\}\) 2

C₃

itemset sup.

\(\{2 3 5\}\) 2

Scan D

L₃

itemset sup.

\(\{2 3 5\}\) 2

Constraint:
Sum\(\{\text{S.price} < ^{2}\text{5}\}\)

The Constrained Apriori Algorithm: Push a Succinct Constraint Deep

Database D

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1 3 4</td>
</tr>
<tr>
<td>200</td>
<td>2 3 5</td>
</tr>
<tr>
<td>300</td>
<td>1 2 3 5</td>
</tr>
<tr>
<td>400</td>
<td>2 5</td>
</tr>
</tbody>
</table>

Scan D

C₁

itemset sup.

\(\{1\}\) 2
\(\{2\}\) 3
\(\{3\}\) 3
\(\{4\}\) 1
\(\{5\}\) 3

L₁

itemset sup.

\(\{1\}\) 2
\(\{2\}\) 3
\(\{3\}\) 3
\(\{5\}\) 3

C₂

itemset sup.

\(\{1 2\}\) 1
\(\{1 3\}\) 2
\(\{1 5\}\) 1
\(\{2 3\}\) 2
\(\{2 5\}\) 3
\(\{3 5\}\) 2

Scan D

L₂

itemset sup.

\(\{1 2\}\) 1
\(\{1 3\}\) 2
\(\{1 5\}\) 1
\(\{2 3\}\) 2
\(\{2 5\}\) 3
\(\{3 5\}\) 2

C₃

itemset sup.

\(\{2 3 5\}\) 2

Scan D

L₃

itemset sup.

\(\{2 3 5\}\) 2

Constraint:
min\(\{\text{S.price} < ^{2}\text{6} 1}\)
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 \(\text{afb}\) violates C
    - So does \(\text{afbh}, \text{afb}^*\)
    - 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. \(\text{avg}(S) \geq v\) 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. \(\text{avg}(S) \leq v\) w.r.t. item value descending order
What Constraints Are Convertible?

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Convertible anti-monotone</th>
<th>Convertible monotone</th>
<th>Strongly convertible</th>
</tr>
</thead>
<tbody>
<tr>
<td>avg(S) ≤ v, ≥ v</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>median(S) ≤ v, ≥ v</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>sum(S) ≤ v (items could be of any value, v ≥ 0)</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>sum(S) ≤ v (items could be of any value, v ≤ 0)</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>sum(S) ≥ v (items could be of any value, v ≥ 0)</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>sum(S) ≥ v (items could be of any value, v ≤ 0)</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

...
Combing Them Together—A General Picture

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Antimonotone</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>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>count($S$) $\leq v$</td>
<td>yes</td>
<td>no</td>
<td>weakly</td>
</tr>
<tr>
<td>count($S$) $\geq v$</td>
<td>no</td>
<td>yes</td>
<td>weakly</td>
</tr>
<tr>
<td>sum($S$) $\leq a$ (a $\in S$, a $\geq 0$)</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>sum($S$) $\geq a$ (a $\in S$, a $\geq 0$)</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>range($S$) $\leq v$</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>range($S$) $\geq v$</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>avg($S$) $\delta v$, $\delta \in {\leq, \geq}$</td>
<td>convertible</td>
<td>convertible</td>
<td>no</td>
</tr>
<tr>
<td>support($S$) $\geq \xi$</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>support($S$) $\leq \xi$</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
</tbody>
</table>

Classification of Constraints

- **Antimonotone**
- **Monotone**
- **Succinct**
- **Convertible anti-monotone**
- **Convertible monotone**
- **Inconvertible**
Mining With Convertible Constraints

- C: \( \text{avg}(S.\text{profit}) \geq 25 \)
- List of items in every transaction in value descending order \( R \):
  - \( \langle a, f, g, d, b, h, c, e \rangle \)
    - \( C \) 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>

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!
Mining With Convertible Constraints

- C: \( \text{avg}(X) \geq 25 \), \( \text{min\_sup}=2 \)
- List 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 TDB once
  - remove infrequent items
    - Item h is dropped
  - Itemsets a and f are good, ...
- Projection-based mining
  - Imposing an appropriate order on item projection
  - Many tough constraints can be converted into (anti-)monotone

<table>
<thead>
<tr>
<th>Item</th>
<th>Value</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>

TDB (min\_sup=2)

<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>

Handling Multiple Constraints

- Different constraints may require different or even conflicting item-ordering
- If there exists an order \( R \) s.t. both \( C_1 \) and \( C_2 \) are convertible w.r.t. \( R \), then there is no conflict between the two convertible constraints
- If there exists conflict on order of items
  - Try to satisfy one constraint first
  - Then using the order for the other constraint to mine frequent itemsets in the corresponding projected database
Mining Association Rules in Large Databases

- Association rule mining
- Algorithms for scalable mining of (single-dimensional Boolean) association rules in transactional databases
- Mining various kinds of association/correlation rules
- Constraint-based association mining
- Sequential pattern mining
- Applications/extensions of frequent pattern mining
- Summary

Sequence Databases and Sequential Pattern Analysis

- Transaction databases, time-series databases vs. sequence databases
- Frequent patterns vs. (frequent) sequential patterns
- Applications of sequential pattern mining
  - Customer shopping sequences:
    - First buy computer, then CD-ROM, and then digital camera, within 3 months.
  - Medical treatment, natural disasters (e.g., earthquakes), science & engineering processes, stocks and markets, etc.
  - Telephone calling patterns, Weblog click 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: \( <(ef)(ab)(df)c>b> \)

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)c&gt;b&gt;</td>
</tr>
<tr>
<td>40</td>
<td>&lt;eg(al)cbbc&gt;</td>
</tr>
</tbody>
</table>

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 on Sequential Pattern Mining

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

• A mining algorithm should
  – find the complete set of patterns, when possible, 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
Studies on Sequential Pattern Mining

- Concept introduction and an initial Apriori-like algorithm
- GSP—An Apriori-based, influential mining method (developed at IBM Almaden)
- From sequential patterns to episodes (Apriori-like + constraints)
- Mining sequential patterns with constraints

A Basic Property of Sequential Patterns: Apriori

- A basic property: Apriori (Agrawal & Sirkant’94)
  - If a sequence S is not frequent
  - Then none of the super-sequences of S is frequent
  - E.g, <hb> is infrequent ⇒ so do <hab> and <(ah)b>

Given support threshold $min\_sup = 2$

<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—A Generalized Sequential Pattern Mining Algorithm

- GSP (Generalized Sequential Pattern) mining algorithm
  - proposed by Agrawal and Srikant, EDBT’96
- 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 Length-1 Sequential Patterns

- Examine GSP using an example
- Initial candidates: all singleton sequences
  - <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>
<th>Sup</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>&lt;bd)cb(ac)</td>
<td>3</td>
</tr>
<tr>
<td>20</td>
<td>&lt;bf)(ce)b(fg)</td>
<td>5</td>
</tr>
<tr>
<td>30</td>
<td>&lt;ah)(bf)abf</td>
<td>4</td>
</tr>
<tr>
<td>40</td>
<td>&lt;be)(ce)d</td>
<td>3</td>
</tr>
<tr>
<td>50</td>
<td>&lt;a(bd)bcb(ade)</td>
<td>2</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

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
</tr>
<tr>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
</tr>
<tr>
<td>f</td>
<td>f</td>
<td>f</td>
<td>f</td>
<td>f</td>
<td>f</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
</tr>
<tr>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
<td>b</td>
</tr>
<tr>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
<td>c</td>
</tr>
<tr>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
<td>d</td>
</tr>
<tr>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
<td>e</td>
</tr>
<tr>
<td>f</td>
<td>f</td>
<td>f</td>
<td>f</td>
<td>f</td>
<td>f</td>
</tr>
</tbody>
</table>

Without Apriori property, \( 8 \times 8 + 8 \times 7 / 2 = 92 \) candidates

Apriori prunes 44.57% candidates

Generating Length-3 Candidates and Finding Length-3 Patterns

- Generate Length-3 Candidates
  - Self-join length-2 sequential patterns
    - Based on the Apriori property
      - \(<ab>, <aa>\) and \(<ba>\) are all length-2 sequential patterns \(\rightarrow <aba>\) is a length-3 candidate
      - \(<bd>, <bb>\) and \(<db>\) are all length-2 sequential patterns \(\rightarrow <bd>b\) 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

5\textsuperscript{th} scan: 1 cand. 1 length-5 seq. pat.
4\textsuperscript{th} scan: 8 cand. 6 length-4 seq. pat.
3\textsuperscript{rd} scan: 46 cand. 19 length-3 seq. pat. 20 cand. not in DB at all
2\textsuperscript{nd} scan: 51 cand. 19 length-2 seq. pat. 10 cand. not in DB at all
1\textsuperscript{st} scan: 8 cand. 6 length-1 seq. pat.

\begin{table}
\begin{tabular}{|c|c|}
\hline
\text{Seq. ID} & \text{Sequence} \\
\hline
10 & \langle bd \rangle \langle cb \rangle \langle ac \rangle \\
20 & \langle bf \rangle \langle ce \rangle \langle bg \rangle \\
30 & \langle ah \rangle \langle bf \rangle \langle abf \rangle \\
40 & \langle be \rangle \langle ce \rangle \langle d \rangle \\
50 & \langle ab \rangle \langle bd \rangle \langle bc \rangle \langle ade \rangle \\
\hline
\end{tabular}
\end{table}

\textit{min_sup} = 2

Bottlenecks of GSP

- A huge set of candidates could be generated
  - 1,000 frequent length-1 sequences generate \(1000 \times 1000 + \frac{1000 \times 999}{2} = 1,499,500\) length-2 candidates!
- Multiple scans of database in mining
- Real challenge: mining long sequential patterns
  - An exponential number of short candidates
  - A length-100 sequential pattern needs \(10^{100}\) candidate sequences!

\[\sum_{i=1}^{100} \binom{100}{i} = 2^{100} - 1 \approx 10^{100}\]
**FreeSpan: Frequent Pattern-Projected Sequential Pattern Mining**

- A divide-and-conquer approach
  - 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
- J. Han, J. Pei, B. Mortazavi-Asi, Q. Chen, U. Dayal, M.C. Hsu, FreeSpan: Frequent pattern-projected sequential pattern mining. In KDD’00.

**Sequence Database SDB**

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; (bd) c b (ac) &gt;</td>
<td>5</td>
</tr>
<tr>
<td>&lt; (bf) (ce) b (fg) &gt;</td>
<td>4</td>
</tr>
<tr>
<td>&lt; (ah) (bf) a b f &gt;</td>
<td>3</td>
</tr>
<tr>
<td>&lt; (be) (ce) d &gt;</td>
<td>3</td>
</tr>
<tr>
<td>&lt; a (bd) b c b (ade) &gt;</td>
<td>2</td>
</tr>
</tbody>
</table>

**f_list**: b:5, c:4, a:3, d:3, e:3, f:2

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*

---

**From FreeSpan to PrefixSpan: Why?**

- 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 do will not shrink much
- PrefixSpan
  - Projection-based
  - But only prefix-based projection: less projections and quickly shrinking sequences
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)c(bc)&gt;</td>
</tr>
</tbody>
</table>
Completeness of PrefixSpan

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)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 <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>

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

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

Optimization Techniques in PrefixSpan

• Physical projection vs. pseudo-projection
  – Pseudo-projection may reduce the effort of projection when the projected database fits in main memory
• Parallel projection vs. partition projection
  – Partition projection may avoid the blowup of disk space
Speed-up by Pseudo-projection

- Major cost of PrefixSpan: projection
  - Postfixes of sequences often appear repeatedly in recursive projected databases
- When (projected) database can be held in main memory, use pointers to form projections
  - Pointer to the sequence
  - Offset of the postfix

\[ s = \langle a(abc)(ac)d(cf) \rangle \]
\[ s|a>: (, 2) \quad \langle abc)(ac)d(cf) \rangle \]
\[ s|ab>: (, 4) \quad \langle c)(ac)d(cf) \rangle \]  

---

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
- However, it is not efficient when database cannot fit in main memory
  - Disk-based random accessing is very costly
- Suggested Approach:
  - Integration of physical and pseudo-projection
  - Swapping to pseudo-projection when the data set fits in memory

---
PrefixSpan Is Faster than GSP and FreeSpan

Effect of Pseudo-Projection
Mining Association Rules in Large Databases

- Association rule mining
- Algorithms for scalable mining of (single-dimensional Boolean) association rules in transactional databases
- Mining various kinds of association/correlation rules
- Constraint-based association mining
- Sequential pattern mining
- Applications/extension of frequent pattern mining
- Summary

Associative Classification

- Mine association possible rules (PR) in form of condset \(\rightarrow c\)
  - Condset: a set of attribute-value pairs
  - C: class label
- Build Classifier
  - Organize rules according to decreasing precedence based on confidence and support
Spatial and Multi-Media Association: A Progressive Refinement Method

- Why progressive refinement?
  - Mining operator can be expensive or cheap, fine or rough
- Superset coverage property:
  - Preserve all the positive answers—allow a positive false test but not a false negative test.
- Two- or multi-step mining:
  - First apply rough/cheap operator (superset coverage)
  - Then apply expensive algorithm on a substantially reduced candidate set (Koperski & Han, SSD’95).

Progressive Refinement
Mining of Spatial Associations

- Hierarchy of spatial relationship:
  - “g_close_to”: near_by, touch, intersect, contain, etc.
  - First search for rough relationship and then refine it.
- Two-step mining of spatial association:
  - Step 1: rough spatial computation (as a filter)
    - Using MBR or R-tree for rough estimation.
  - Step2: Detailed spatial algorithm (as refinement)
    - Apply only to those objects which have passed the rough spatial association test (no less than min_support)
Mining Multimedia Associations

Correlations with color, spatial relationships, etc. From coarse to Fine Resolution mining

Further Evolution of PrefixSpan

- Closed- and max- sequential patterns
  - Finding only the most meaningful (longest) sequential patterns
- Constraint-based sequential pattern growth
  - Adding user-specific constraints
- From sequential patterns to structured patterns
  - Beyond sequential patterns, mining structured patterns in XML documents
Closed- and Max- Sequential Patterns

• A closed- sequential pattern is a frequent sequence s where there is no proper super-sequence of s sharing the same support count with s.

• A max- sequential pattern is a sequential pattern p s.t. any proper super-pattern of p is not frequent.

• Benefit of the notion of closed sequential patterns:
  - \(<a_1, a_2 \ldots a_{50}>, \ <a_1, a_2 \ldots a_{100}>\), with min_sup = 1
  - There are 2^{100} sequential patterns, but only 2 are closed.

• Similar benefits for the notion of max- sequential-patterns.

Methods for Mining Closed- and Max- Sequential Patterns

• PrefixSpan or FreeSpan can be viewed as projection-guided depth-first search.

• For mining max- sequential patterns, any sequence which does not contain anything beyond the already discovered ones will be removed from the projected DB:
  - \(<a_1, a_2 \ldots a_{50}>, \ <a_1, a_2 \ldots a_{100}>\), with min_sup = 1
  - If we have found a max-sequential pattern \(<a_1, a_2 \ldots a_{100}>\), nothing will be projected in any projected DB.

• Similar ideas can be applied for mining closed-sequential-patterns.
Constraint-Based Sequential Pattern Mining

- Constraint-based sequential pattern mining
  - Constraints: User-specified, for focused mining of desired patterns
  - How to explore efficient mining with constraints? — Optimization
- Classification of constraints
  - Anti-monotone: E.g., value_sum(S) < 150, min(S) > 10
  - Monotone: E.g., count(S) > 5, S ⊆ {PC, digital_camera}
  - Succinct: E.g., length(S) ≥ 10, S ⊆ {Pentium, MS/Office, MS/Money}
  - Convertible: E.g., value_avg(S) < 25, profit_sum(S) > 160, max(S)/avg(S) < 2, median(S) – min(S) > 5
  - Inconvertible: E.g., avg(S) – median(S) = 0

Sequential Pattern Growth for Constraint-Based Mining

- Efficient mining with convertible constraints
  - Not solvable by candidate generation-and-test methodology
  - Easily push-able into the sequential pattern growth framework
- Example: push avg(S) < 25 in frequent pattern growth
  - project items in value (price/profit depending on mining semantics) ascending/descending order for sequential pattern growth
  - Grow each pattern by sequential pattern growth
  - If avg(current_pattern) ≤ 25, toss the current_pattern
    - Why?—future growths always make it bigger
    - But why not candidate generation?—no structure or ordering in growth
From Sequential Patterns to Structured Patterns

- Sets, sequences, trees and other structures
  - Transaction DB: Sets of items
    - \{\{l_1, l_2, \ldots, l_m\}, \ldots\}
  - Seq. DB: Sequences of sets:
    - \{<l_1, l_2>, \ldots, <l_m, l_n>, \ldots\}
  - Sets of Sequences:
    - \{\{<l_1, l_2>, \ldots, <l_m, l_n>, \ldots\}, \ldots\}
  - Sets of trees (each element being a tree):
    - \{t_1, t_2, \ldots, t_n\}
- Applications: Mining structured patterns in XML documents

Mining Association Rules in Large Databases

- Association rule mining
- Algorithms for scalable mining of (single-dimensional Boolean) association rules in transactional databases
- Mining various kinds of association/correlation rules
- Constraint-based association mining
- Sequential pattern mining
- Applications/extensions of frequent pattern mining
- Summary
Frequent-Pattern Mining: Achievements

- Frequent pattern mining—an important task in data mining
- Frequent pattern mining methodology
  - Candidate generation & test vs. projection-based (frequent-pattern growth)
  - Vertical vs. horizontal format
  - Various optimization methods: database partition, scan reduction, hash tree, sampling, border computation, clustering, etc.
- Related frequent-pattern mining algorithm: scope extension
  - Mining closed frequent itemsets and max-patterns (e.g., MaxMiner, CLOSET, CHARM, etc.)
  - Mining multi-level, multi-dimensional frequent patterns with flexible support constraints
  - Constraint pushing for mining optimization
  - From frequent patterns to correlation and causality

Frequent-Pattern Mining: Applications

- Related problems which need frequent pattern mining
  - Association-based classification
  - Iceberg cube computation
  - Database compression by fascicles and frequent patterns
  - Mining sequential patterns (GSP, PrefixSpan, SPADE, etc.)
  - Mining partial periodicity, cyclic associations, etc.
  - Mining frequent structures, trends, etc.
- Typical application examples
  - Market-basket analysis, Weblog analysis, DNA mining, etc.
### Frequent-Pattern Mining: Research Problems

- Multi-dimensional gradient analysis: patterns regarding changes and differences
  - Not just counts—other measures, e.g., avg(profit)
- Mining top-k frequent patterns without support constraint
- Mining fault-tolerant associations
  - “3 out of 4 courses excellent” leads to A in data mining
- Fascicles and database compression by frequent pattern mining
- Partial periodic patterns
- DNA sequence analysis and pattern classification

### References: Frequent-pattern Mining Methods

- J. Han, J. Pei, and Y. Yin: “Mining frequent patterns without candidate generation”. In Proc. ACM-SIGMOD’2000, pp. 1-12, Dallas, TX, May 2000.
## References: Frequent-pattern Mining Methods

- R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. SIGMOD'96, 1-12, Montreal, Canada.

## References: Frequent-pattern Mining (Performance Improvements)

- D.W. Cheung, J. Han, V. Ng, and C.Y. Wong. Maintenance of discovered association rules in large databases: An incremental updating technique. ICDE'96, New Orleans, LA.
- E.-H. Han, G. Karypis, and V. Kumar. Scalable parallel data mining for association rules. SIGMOD'97, Tucson, Arizona.
References: Frequent-pattern Mining (Performance Improvements)

- J.S. Park, M.S. Chen, and P.S. Yu. An effective hash-based algorithm for mining association rules. SIGMOD’95, San Jose, CA.

References: Frequent-pattern Mining (Multi-level, correlation, ratio rules, etc.)

- J. Han and Y. Fu. Discovery of multiple-level association rules from large databases. VLDB’95, 420-431, Zurich, Switzerland.
- J. Pei, A. K. H. Tung, J. Han. Fault-Tolerant Frequent Pattern Mining: Problems and Challenges. SIGMOD DMKD’01, Santa Barbara, CA.
References: Mining Max-patterns and Closed itemsets

- J. Pei, J. Han, and R. Mao, "CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets", Proc. 2000 ACM-SIGMOD Int. Workshop on Data Mining and Knowledge Discovery (DMKD’00), Dallas, TX, May 2000.
- M. Zaki. CHARM: An Efficient Algorithm for Closed Association Rule Mining, SIAM’02

References: Constraint-base Frequent-pattern Mining

- J. Han, L. V. S. Lakshmanan, and R. T. Ng, "Constraint-Based, Multidimensional Data Mining", COMPUTER (special issues on Data Mining), 32(8): 46-50, 1999.
- L. V. S. Lakshmanan, R. Ng, J. Han and A. Pang. "Optimization of Constrained Frequent Set Queries with 2-Variable Constraints", SIGMOD’99
- R. Ng, L.V.S. Lakshmanan, J. Han & A. Pang. "Exploratory mining and pruning optimizations of constrained association rules." SIGMOD’98
- J. Pei, J. Han, and L. V. S. Lakshmanan, "Mining Frequent Itemsets with Convertible Constraints", Proc. 2001 Int. Conf. on Data Engineering (ICDE’01), April 2001.
- J. Pei and J. Han "Can We Push More Constraints into Frequent Pattern Mining?", Proc. 2000 Int. Conf. on Knowledge Discovery and Data Mining (KDD’00), Boston, MA. August 2000.
- R. Srikanth, Q. Vu, and R. Agrawal. Mining association rules with item constraints. KDD’97, 67-73, Newport Beach, California
References: Sequential Pattern Mining Methods

- R. Agrawal and R. Srikant. Mining sequential patterns. ICDE’95, 3-14, Taipei, Taiwan.
- J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, M.-C. Hsu, "FreeSpan: Frequent Pattern-Projected Sequential Pattern Mining", Proc. 2000 Int. Conf. on Knowledge Discovery and Data Mining (KDD’00), Boston, MA, August 2000.

References: Sequential Pattern Mining Methods

- J. Pei, J. Han, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu, "PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth", Proc. 2001 Int. Conf. on Data Engineering (ICDE’01), Heidelberg, Germany, April 2001.
- B. Ozden, S. Ramaswamy, and A. Silberschatz. Cyclic association rules. ICDE’98, 412-421, Orlando, FL.
References: Frequent-pattern Mining in Spatial, Multimedia, Text & Web Databases

- A. K. H. Tung, H. Lu, J. Han, and L. Feng, "Breaking the Barrier of Transactions: Mining Inter-Transaction Association Rules", Proc. 1999 Int. Conf. on Knowledge Discovery and Data Mining (KDD'99), San Diego, CA, Aug. 1999, pp. 297-301.
- J. Han, G. Dong and Y. Yin, "Efficient Mining of Partial Periodic Patterns in Time Series Database", Proc. 1999 Int. Conf. on Data Engineering (ICDE'99), Sydney, Australia, March 1999, pp. 106-115
- O. R. Zaiane, M. Xin, J. Han, "Discovering Web Access Patterns and Trends by Applying OLAP and Data Mining Technology on Web Logs", Proc. Advances in Digital Libraries Conf. (ADL'98), Santa Barbara, CA, April 1998, pp. 19-29

References: Frequent-pattern Mining for Classification and Data Cube Computation

- M. Kamber, J. Han, and J. Y. Chiang, Metarule-guided mining of multi-dimensional association rules using data cubes. KDD'97, 207-210, Newport Beach, California.
- K. Beyer and R. Ramakrishnan, Bottom-up computation of sparse and iceberg cubes. SIGMOD'99