Chapter 7: Advanced Frequent Pattern Mining

- Mining Diverse Patterns
- Sequential Pattern Mining
- Constraint-Based Frequent Pattern Mining
- Graph Pattern Mining
- Pattern Mining Application: Mining Software Copy-and-Paste Bugs
- Summary
Mining Diverse Patterns

- Mining Multiple-Level Associations
- Mining Multi-Dimensional Associations
- Mining Quantitative Associations
- Mining Negative Correlations
- Mining Compressed and Redundancy-Aware Patterns
Mining Multiple-Level Frequent Patterns

- Items often form hierarchies
  - Ex.: Dairyland 2% milk; Wonder wheat bread
- How to set min-support thresholds?
  - Uniform min-support across multiple levels (reasonable?)
  - Level-reduced min-support: Items at the lower level are expected to have lower support
- Efficient mining: *Shared* multi-level mining
  - Use the lowest min-support to pass down the set of candidates

**Uniform support**

- Milk
  - Level 1
    - min_sup = 5%
  - [support = 10%]
- 2% Milk
  - Level 2
    - min_sup = 5%
  - [support = 6%]
- Skim Milk
  - Level 2
    - min_sup = 1%
  - [support = 2%]

**Reduced support**

- Level 1
  - min_sup = 5%
Redundancy Filtering at Mining Multi-Level Associations

- Multi-level association mining may generate many redundant rules
- Redundancy filtering: Some rules may be redundant due to “ancestor” relationships between items
  - milk ⇒ wheat bread [support = 8%, confidence = 70%] (1)
  - 2% milk ⇒ wheat bread [support = 2%, confidence = 72%] (2)
- Suppose the 2% milk sold is about ¼ of milk sold in gallons
  - (2) should be able to be “derived” from (1)
- A rule is *redundant* if its support is close to the “expected” value, according to its “ancestor” rule, and it has a similar confidence as its “ancestor”
- Rule (1) is an ancestor of rule (2), which one to prune?
Customized Min-Supports for Different Kinds of Items

- We have used the same min-support threshold for all the items or item sets to be mined in each association mining.
- In reality, some items (e.g., diamond, watch, ...) are valuable but less frequent.
- It is necessary to have customized min-support settings for different kinds of items.
- One Method: Use group-based “individualized” min-support
  - E.g., {diamond, watch}: 0.05%; {bread, milk}: 5%; ...
  - How to mine such rules efficiently?
    - Existing scalable mining algorithms can be easily extended to cover such cases.
Mining Multi-Dimensional Associations

- Single-dimensional rules (e.g., items are all in “product” dimension)
  - buys(X, “milk”) ⇒ buys(X, “bread”)

- Multi-dimensional rules (i.e., items in ≥2 dimensions or predicates)
  - Inter-dimension association rules (*no repeated predicates*)
  - Hybrid-dimension association rules (*repeated predicates*)

- Attributes can be categorical or numerical
  - Categorical Attributes (e.g., *profession, product*: no ordering among values): Data cube for inter-dimension association
  - Quantitative Attributes: Numeric, implicit ordering among values—discretization, clustering, and gradient approaches
Mining Quantitative Associations

- Mining associations with numerical attributes
  - Ex.: Numerical attributes: age and salary

- Methods
  - Static discretization based on predefined concept hierarchies
    - Discretization on each dimension with hierarchy
      - age: {0-10, 10-20, ..., 90-100} → {young, mid-aged, old}
  - Dynamic discretization based on data distribution
  - Clustering: Distance-based association
    - First one-dimensional clustering, then association
  - Deviation analysis:
    - Gender = female ⇒ Wage: mean=$7/hr (overall mean = $9)
Mining Extraordinary Phenomena in Quantitative Association Mining

- Mining extraordinary (i.e., interesting) phenomena
  - Ex.: Gender = female ⇒ Wage: mean=$7/hr (overall mean = $9)
  - LHS: a subset of the population
  - RHS: an extraordinary behavior of this subset
- The rule is accepted only if a statistical test (e.g., Z-test) confirms the inference with high confidence
- Subrule: Highlights the extraordinary behavior of a subset of the population of the super rule
  - Ex.: (Gender = female) ^ (South = yes) ⇒ mean wage = $6.3/hr
- Rule condition can be categorical or numerical (quantitative rules)
  - Ex.: Education in [14-18] (yrs) ⇒ mean wage = $11.64/hr
- Efficient methods have been developed for mining such extraordinary rules (e.g., Aumann and Lindell@KDD’99)
Rare Patterns vs. Negative Patterns

- Rare patterns
  - Very low support but interesting (e.g., buying Rolex watches)
  - How to mine them? Setting individualized, group-based min-support thresholds for different groups of items

- Negative patterns
  - Negatively correlated: Unlikely to happen together
  - Ex.: Since it is unlikely that the same customer buys both a Ford Expedition (an SUV car) and a Ford Fusion (a hybrid car), buying a Ford Expedition and buying a Ford Fusion are likely negatively correlated patterns
  - How to define negative patterns?
Defining Negative Correlated Patterns

- A support-based definition
  - If itemsets A and B are both frequent but rarely occur together, i.e., \( \text{sup}(A \cup B) \ll \text{sup}(A) \times \text{sup}(B) \)
  - Then A and B are negatively correlated

- Is this a good definition for large transaction datasets?

- Ex.: Suppose a store sold two needle packages A and B 100 times each, but only one transaction contained both A and B
  - When there are in total 200 transactions, we have
    - \( s(A \cup B) = 0.005, s(A) \times s(B) = 0.25, s(A \cup B) \ll s(A) \times s(B) \)
  - But when there are \( 10^5 \) transactions, we have
    - \( s(A \cup B) = 1/10^5, s(A) \times s(B) = 1/10^3 \times 1/10^3, s(A \cup B) > s(A) \times s(B) \)

- What is the problem? — Null transactions: The support-based definition is not null-invariant!
Defining Negative Correlation: Need Null-Invariance in Definition

- A good definition on negative correlation should take care of the null-invariance problem
- Whether two itemsets A and B are negatively correlated should not be influenced by the number of null-transactions
- A Kulczynski measure-based definition
  - If itemsets A and B are frequent but
    \[(\frac{s(A \cup B)}{s(A)} + \frac{s(A \cup B)}{s(B)})/2 < \epsilon,\]
    where \(\epsilon\) is a negative pattern threshold, then A and B are negatively correlated
- For the same needle package problem:
  - No matter there are in total 200 or \(10^5\) transactions
  - If \(\epsilon = 0.01\), we have
    \[(\frac{s(A \cup B)}{s(A)} + \frac{s(A \cup B)}{s(B)})/2 = (0.01 + 0.01)/2 < \epsilon\]
Mining Compressed Patterns

- Why mining compressed patterns?
  - Too many scattered patterns but not so meaningful

- Pattern distance measure
  \[ Dist(P_1, P_2) = 1 - \frac{|T(P_1) \cap T(P_2)|}{|T(P_1) \cup T(P_2)|} \]

- \(\delta\)-clustering: For each pattern \(P\), find all patterns which can be expressed by \(P\) and whose distance to \(P\) is within \(\delta\) (\(\delta\)-cover)

- All patterns in the cluster can be represented by \(P\)

- Method for efficient, direct mining of compressed frequent patterns (e.g., D. Xin, J. Han, X. Yan, H. Cheng, "On Compressing Frequent Patterns", Knowledge and Data Engineering, 60:5-29, 2007)

<table>
<thead>
<tr>
<th>Pat-ID</th>
<th>Item-Sets</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>{38,16,18,12}</td>
<td>205227</td>
</tr>
<tr>
<td>P2</td>
<td>{38,16,18,12,17}</td>
<td>205211</td>
</tr>
<tr>
<td>P3</td>
<td>{39,38,16,18,12,17}</td>
<td>101758</td>
</tr>
<tr>
<td>P4</td>
<td>{39,16,18,12,17}</td>
<td>161563</td>
</tr>
<tr>
<td>P5</td>
<td>{39,16,18,12}</td>
<td>161576</td>
</tr>
</tbody>
</table>

- Closed patterns
  - P1, P2, P3, P4, P5
  - Emphasizes too much on support
  - There is no compression

- Max-patterns
  - P3: information loss

- Desired output (a good balance):
  - P2, P3, P4
Desired patterns: high significance & low redundancy

- Method: Use MMS (Maximal Marginal Significance) for measuring the combined significance of a pattern set
- Xin et al., Extracting Redundancy-Aware Top-K Patterns, KDD’06
Chapter 7: Advanced Frequent Pattern Mining

- Mining Diverse Patterns
- Sequential Pattern Mining
- Constraint-Based Frequent Pattern Mining
- Graph Pattern Mining
- Pattern Mining Application: Mining Software Copy-and-Paste Bugs
- Summary
Sequential Pattern Mining

- Sequential Pattern and Sequential Pattern Mining
- GSP: Apriori-Based Sequential Pattern Mining
- SPADE: Sequential Pattern Mining in Vertical Data Format
- PrefixSpan: Sequential Pattern Mining by Pattern-Growth
- CloSpan: Mining Closed Sequential Patterns
Sequential Databases & Sequential Patterns

- Sequential pattern mining has broad applications
  - Customer shopping sequences
    - Purchase a laptop first, then a digital camera, and then a smartphone, within 6 months
  - Medical treatments, natural disasters (e.g., earthquakes), science & engineering processes, stocks and markets, ...
  - Weblog click streams, calling patterns, ...
  - Software engineering: Program execution sequences, ...
  - Biological sequences: DNA, protein, ...
- Transaction DB, sequence DB vs. time-series DB
- Gapped vs. non-gapped sequential patterns
  - Shopping sequences, clicking streams vs. biological sequences
Sequential Pattern and Sequential Pattern Mining

- **Sequential pattern mining**: Given a set of sequences, find the complete set of frequent subsequences (i.e., satisfying the min_sup threshold)

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

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

- An **element** may contain a set of **items** (also called **events**)
- Items within an element are unordered and we list them alphabetically

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

- Given **support threshold** min_sup = 2, <(ab)c> is a **sequential pattern**
Sequential Pattern Mining Algorithms

- Algorithm requirement: Efficient, scalable, finding complete set, incorporating various kinds of user-specific constraints

- The Apriori property still holds: If a subsequence \( s_1 \) is infrequent, none of \( s_1 \)'s super-sequences can be frequent

- Representative algorithms
  - GSP (Generalized Sequential Patterns): Srikant & Agrawal @ EDBT’96)
  - Vertical format-based mining: SPADE (Zaki@Machine Learning’00)
  - Pattern-growth methods: PrefixSpan (Pei, et al. @TKDE’04)
  - Mining closed sequential patterns: CloSpan (Yan, et al. @SDM’03)
  - Constraint-based sequential pattern mining (to be covered in the constraint mining section)
GSP: Apriori-Based Sequential Pattern Mining

- Initial candidates: All 8-singleton sequences
  - \(<a>, <b>, <c>, <d>, <e>, <f>, <g>, <h>\)
- Scan DB once, count support for each candidate
- Generate length-2 candidate sequences

\[
\begin{array}{|c|c|c|c|c|c|}
\hline
\text{Cand.} & \text{sup} \\
\hline
\langle a \rangle & 3 \\
\langle b \rangle & 5 \\
\langle c \rangle & 4 \\
\langle d \rangle & 3 \\
\langle e \rangle & 3 \\
\langle f \rangle & 2 \\
\langle g \rangle & 1 \\
\langle h \rangle & 1 \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|c|c|c|}
\hline
\langle a \rangle & \langle a \rangle & \langle ab \rangle & \langle ac \rangle & \langle ad \rangle & \langle ae \rangle & \langle af \rangle \\
\langle b \rangle & \langle ba \rangle & \langle bb \rangle & \langle bc \rangle & \langle bd \rangle & \langle be \rangle & \langle bf \rangle \\
\langle c \rangle & \langle ca \rangle & \langle cb \rangle & \langle cc \rangle & \langle cd \rangle & \langle ce \rangle & \langle cf \rangle \\
\langle d \rangle & \langle da \rangle & \langle db \rangle & \langle dc \rangle & \langle dd \rangle & \langle de \rangle & \langle df \rangle \\
\langle e \rangle & \langle ea \rangle & \langle eb \rangle & \langle ec \rangle & \langle ed \rangle & \langle ee \rangle & \langle ef \rangle \\
\langle f \rangle & \langle fa \rangle & \langle fb \rangle & \langle fc \rangle & \langle fd \rangle & \langle fe \rangle & \langle ff \rangle \\
\hline
\end{array}
\]

- \(\text{min}_\text{sup} = 2\)

Without Apriori pruning:
- (8 singletons) \(8 \times 8 + 8 \times 7 / 2 = 92\) length-2 candidates

With pruning, length-2 candidates: \(36 + 15 = 51\)

GSP (Generalized Sequential Patterns): Srikant & Agrawal @ EDBT’96
GSP Mining and Pruning

5th scan: 1 cand. 1 length-5 seq. pat.
4th scan: 8 cand. 7 length-4 seq. pat.
3rd scan: 46 cand. 20 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.

- Repeat (for each level (i.e., length-k))
  - Scan DB to find length-k frequent sequences
  - Generate length-(k+1) candidate sequences from length-k frequent sequences using Apriori
  - set k = k+1
  - Until no frequent sequence or no candidate can be found

<table>
<thead>
<tr>
<th>SID</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>
Sequential Pattern Mining in Vertical Data Format: The SPADE Algorithm

- A sequence database is mapped to: <SID, EID>
- Grow the subsequences (patterns) one item at a time by Apriori candidate generation

Ref: SPADE (Sequential PAttern Discovery using Equivalent Class) [M. Zaki 2001]
PrefixSpan: A Pattern-Growth Approach

- PrefixSpan Mining: Prefix Projections
  - Step 1: Find length-1 sequential patterns
    - <a>, <b>, <c>, <d>, <e>, <f>
  - Step 2: Divide search space and mine each projected DB
    - <a>-projected DB,
    - <b>-projected DB,
    - ...
    - <f>-projected DB, ...

Prefix and suffix
- Given <a(abc)(ac)d(cf)>
- Prefixes: <a>, <aa>, <a(ab)>, <a(abc)>, ...
- Suffix: Prefixes-based projection

<table>
<thead>
<tr>
<th>SID</th>
<th>Sequence</th>
<th>Prefix</th>
<th>Suffix (Projection)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>&lt;a(abc)(ac)d(cf)&gt;</td>
<td>&lt;a&gt;</td>
<td>&lt;a(abc)(ac)d(cf)&gt;</td>
</tr>
<tr>
<td>20</td>
<td>&lt;(ad)c(bc)(ae)&gt;</td>
<td>&lt;aa&gt;</td>
<td>&lt;(_bc)(ac)d(cf)&gt;</td>
</tr>
<tr>
<td>30</td>
<td>&lt;(ef)(ab)(df)cb&gt;</td>
<td>&lt;ab&gt;</td>
<td>&lt;(_c)(ac)d(cf)&gt;</td>
</tr>
<tr>
<td>40</td>
<td>&lt;eg(af)cbbc&gt;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PrefixSpan (Prefix-projected Sequential pattern mining)
Pei, et al. @TKDE’04

min_sup = 2
PrefixSpan: Mining Prefix-Projected DBs

<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>

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

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

PrefixSpan:
- No candidate subsequences to be generated
- Projected DBs keep shrinking

Major strength of PrefixSpan:
- Maximal length = 10
- No candidate subsequences to be generated
- Projected DBs keep shrinking

$\text{min_sup} = 2$
Implementation Consideration: Pseudo-Projection vs. Physical Projection

- Major cost of PrefixSpan: Constructing projected DBs
- Suffixes largely repeating in recursive projected DBs
- When DB can be held in main memory, use pseudo projection
  - No physically copying suffixes
  - Pointer to the sequence
  - Offset of the suffix
- But if it does not fit in memory
  - Physical projection
- Suggested approach:
  - Integration of physical and pseudo-projection
  - Swapping to pseudo-projection when the data fits in memory
A closed sequential pattern $s$: There exists no superpattern $s'$ such that $s' \supset s$, and $s'$ and $s$ have the same support

Which ones are closed? $\langle abc \rangle: 20$, $\langle abcd \rangle: 20$, $\langle abcde \rangle: 15$

Why directly mine closed sequential patterns?
- Reduce # of (redundant) patterns
- Attain the same expressive power

Property $P_1$: If $s \supset s_1$, $s$ is closed iff two project DBs have the same size

Explore Backward Subpattern and Backward Superpattern pruning to prune redundant search space

Greatly enhances efficiency (Yan, et al., SDM’03)
CloSpan: When Two Projected DBs Have the Same Size

- If \( s \supseteq s_2 \), \( s \) is closed iff two project DBs have the same size
- When two projected sequence DBs have the same size?
- Here is one example:

<table>
<thead>
<tr>
<th>ID</th>
<th>Sequence</th>
<th>( \min_sup = 2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>&lt;aeefbcg&gt;</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>&lt;afegb(ac)&gt;</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>&lt;(af)ea&gt;</td>
<td></td>
</tr>
</tbody>
</table>

- Only need to keep size = 12 (including parentheses)
- Backward subpattern pruning
- Backward superpattern pruning
Chapter 7: Advanced Frequent Pattern Mining

- Mining Diverse Patterns
- Sequential Pattern Mining
- Constraint-Based Frequent Pattern Mining
- Graph Pattern Mining
- Pattern Mining Application: Mining Software Copy-and-Paste Bugs
- Summary
Constraint-Based Pattern Mining

- Why Constraint-Based Mining?
- Different Kinds of Constraints: Different Pruning Strategies
- Constrained Mining with Pattern Anti-Monotonicity
- Constrained Mining with Pattern Monotonicity
- Constrained Mining with Data Anti-Monotonicity
- Constrained Mining with Succinct Constraints
- Constrained Mining with Convertible Constraints
- Handling Multiple Constraints
- Constraint-Based Sequential-Pattern Mining
Why Constraint-Based Mining?

- Finding **all** the patterns in a dataset **autonomously**?—unrealistic!
- Too many patterns but not necessarily user-interested!
- Pattern mining in practice: Often a user-guided, **interactive** process
  - User directs what to be mined using a **data mining query language** (or a graphical user interface), **specifying various kinds of constraints**
- What is constraint-based mining?
  - Mine together with user-provided constraints
- Why constraint-based mining?
  - User flexibility: User provides **constraints** on what to be mined
  - Optimization: System explores such constraints for mining efficiency
  - E.g., Push constraints deeply into the mining process
Various Kinds of User-Specified Constraints in Data Mining

- **Knowledge type constraint**—Specifying what kinds of knowledge to mine
  - Ex.: Classification, association, clustering, outlier finding, ...

- **Data constraint**—using SQL-like queries
  - Ex.: Find products sold together in NY stores this year

- **Dimension/level constraint**—similar to projection in relational database
  - Ex.: In relevance to region, price, brand, customer category

- **Interestingness constraint**—various kinds of thresholds
  - Ex.: Strong rules: $\text{min\_sup} \geq 0.02$, $\text{min\_conf} \geq 0.6$, $\text{min\_correlation} \geq 0.7$

- **Rule (or pattern) constraint**
  - Ex.: Small sales (price < $10) triggers big sales (sum > $200)

The focus of this study
A constraint $c$ is **anti-monotone**

- If an itemset $S$ violates constraint $c$, so does any of its superset
- That is, mining on itemset $S$ can be terminated

- **Ex. 1:** $c_1$: $\sum(S . price) \leq v$ is anti-monotone
- **Ex. 2:** $c_2$: range($S . profit$) $\leq 15$ is anti-monotone

- Itemset $ab$ violates $c_2$ (range($ab$) = 40)
- So does every superset of $ab$

- **Ex. 3:** $c_3$: $\sum(S . Price) \geq v$ is **not** anti-monotone
- **Ex. 4.** Is $c_4$: $support(S) \geq \sigma$ anti-monotone?

Yes! Apriori pruning is essentially pruning with an anti-monotonic constraint!

Note: item.price > 0
Profit can be negative
Pattern Monotonicity and Its Roles

- A constraint $c$ is **monotone**: If an itemset $S$ **satisfies** the constraint $c$, so does any of its superset.
  - That is, we do not need to check $c$ in subsequent mining.
  - Ex. 1: $c_1: \text{sum}(S.Price) \geq v$ is monotone.
  - Ex. 2: $c_2: \text{min}(S.Price) \leq v$ is monotone.
  - Ex. 3: $c_3: \text{range}(S.profit) \geq 15$ is monotone.
  - Itemset $ab$ satisfies $c_3$.
  - 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, h</td>
</tr>
<tr>
<td>20</td>
<td>b, c, d, f, g, h</td>
</tr>
<tr>
<td>30</td>
<td>b, c, d, f, g</td>
</tr>
<tr>
<td>40</td>
<td>a, c, e, f, g</td>
</tr>
<tr>
<td></td>
<td><strong>min_sup = 2</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Item</th>
<th>Price</th>
<th>Profit</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>100</td>
<td>40</td>
</tr>
<tr>
<td>b</td>
<td>40</td>
<td>0</td>
</tr>
<tr>
<td>c</td>
<td>150</td>
<td>-20</td>
</tr>
<tr>
<td>d</td>
<td>35</td>
<td>-15</td>
</tr>
<tr>
<td>e</td>
<td>55</td>
<td>-30</td>
</tr>
<tr>
<td>f</td>
<td>45</td>
<td>-10</td>
</tr>
<tr>
<td>g</td>
<td>80</td>
<td>20</td>
</tr>
<tr>
<td>h</td>
<td>10</td>
<td>5</td>
</tr>
</tbody>
</table>

Note: item.price > 0
Profit can be negative
A constraint $c$ is **data anti-monotone**: In the mining process, if a data entry $t$ cannot satisfy a pattern $p$ under $c$, $t$ cannot satisfy $p$'s superset either.

- **Data space pruning**: Data entry $t$ can be pruned.
  - **Ex. 1**: $c_1$: \( \sum (S.\text{Profit}) \geq \nu \) is **data anti-monotone**
  - Let constraint $c_1$ be: \( \sum (S.\text{Profit}) \geq 25 \)
    - $T_{30}$: \{b, c, d, f, g\} can be removed since none of their combinations can make an $S$ whose sum of the profit is $\geq 25$
  - **Ex. 2**: $c_2$: \( \min (S.\text{Price}) \leq \nu \) is **data anti-monotone**
    - Consider $\nu = 5$ but every item in a transaction, say $T_{50}$, has a price higher than 10
  - **Ex. 3**: $c_3$: \( \text{range}(S.\text{Profit}) > 25 \) is **data anti-monotone**

Note: item.price > 0
Profit can be negative
Data Space Pruning Should Be Explored Recursively

- Example. $c_3$: $\text{range}(S.\text{Profit}) > 25$
  - We check $b$’s projected database
  - But item “a” is infrequent ($\text{sup} = 1$)
  - After removing “a (40)” from $T_{10}$
  - $T_{10}$ cannot satisfy $c_3$ any more
  - Since “b (0)” and “c (-20), d (-15), f (-10), h (5)”
  - By removing $T_{10}$, we can also prune “h” in $T_{20}$

- Note: $c_3$ prunes $T_{10}$ effectively only after “a” is pruned (by min-sup) in $b$’s projected DB
Succinctness: Pruning Both Data and Pattern Spaces

- Succinctness: If the constraint $c$ can be enforced by directly manipulating the data
- Ex. 1: To find those patterns without item $i$
  - Remove $i$ from DB and then mine (pattern space pruning)
- Ex. 2: To find those patterns containing item $i$
  - Mine only $i$-projected DB (data space pruning)
- Ex. 3: $c_3$: $\min(S.Price) \leq v$ is succinct
  - Start with only items whose price $\leq v$ and remove transactions with high-price items only (pattern + data space pruning)
- Ex. 4: $c_4$: $\sum(S.Price) \geq v$ is not succinct
  - It cannot be determined beforehand since sum of the price of itemset $S$ keeps increasing
Convertible Constraints: Ordering Data in Transactions

- Convert tough constraints into (anti-)monotone by proper ordering of items in transactions
- Examine $c_1$: $\text{avg}(S.\text{profit}) > 20$
  - Order items in (profit) value-descending order
  - $<a, g, f, b, h, d, c, e>$
  - An itemset $ab$ violates $c_1$ ($\text{avg}(ab) = 20$)
  - So does $ab^*$ (i.e., $ab$-projected DB)
  - $C_1$: anti-monotone if patterns grow in the right order!
- Can item-reordering work for Apriori?
  - Level-wise candidate generation requires multi-way checking!
  - $\text{avg}(agf) = 21.7 > 20$, but $\text{avg}(gf) = 12.5 < 20$
  - Apriori will not generate “agf” as a candidate

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

<table>
<thead>
<tr>
<th>min_sup = 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>a</td>
</tr>
<tr>
<td>b</td>
</tr>
<tr>
<td>c</td>
</tr>
<tr>
<td>d</td>
</tr>
<tr>
<td>e</td>
</tr>
<tr>
<td>f</td>
</tr>
<tr>
<td>g</td>
</tr>
<tr>
<td>h</td>
</tr>
</tbody>
</table>
Different Kinds of Constraints Lead to Different Pruning Strategies

- In summary, constraints can be categorized as
  - **Pattern space pruning constraints** vs. **data space pruning constraints**
  - **Pattern space pruning constraints**
    - **Anti-monotonic**: If constraint $c$ is violated, its further mining can be terminated
    - **Monotonic**: If $c$ is satisfied, no need to check $c$ again
    - **Succinct**: If the constraint $c$ can be enforced by directly manipulating the data
    - **Convertible**: $c$ can be converted to monotonic or anti-monotonic if items can be properly ordered in processing
  - **Data space pruning constraints**
    - **Data succinct**: Data space can be pruned at the initial pattern mining process
    - **Data anti-monotonic**: If a transaction $t$ does not satisfy $c$, then $t$ can be pruned to reduce data processing effort
How to Handle Multiple Constraints?

- It is beneficial to use multiple constraints in pattern mining
- But different constraints may require potentially conflicting item-ordering
  - If there exists conflict ordering between $c_1$ and $c_2$
    - Try to sort data and enforce *one constraint* first (which one?)
    - Then enforce the other constraint when mining the projected databases
- Ex. $c_1$: $\text{avg}(S.\text{profit}) > 20$, and $c_2$: $\text{avg}(S.\text{price}) < 50$
  - Assume $c_1$ has more pruning power
  - Sort in profit descending order and use $c_1$ first
  - For each project DB, sort trans. in price ascending order and use $c_2$ at mining
Constraint-Based Sequential-Pattern Mining

- Share many similarities with constraint-based itemset mining

- **Anti-monotonic:** If S violates $c$, the super-sequences of S also violate $c$
  - $\text{sum}(S.\text{price}) < 150; \text{min}(S.\text{value}) > 10$

- **Monotonic:** If S satisfies $c$, the super-sequences of S also do so
  - $\text{element_count}(S) > 5; S \supseteq \{\text{PC, digital_camera}\}$

- **Data anti-monotonic:** If a sequence $s_1$ with respect to S violates $c_3$, $s_1$ can be removed
  - $c_3: \text{sum}(S.\text{price}) \geq v$

- **Succinct:** Enforce constraint $c$ by explicitly manipulating data
  - $S \supseteq \{\text{i-phone, MacAir}\}$

- **Convertible:** Projection based on the sorted value not sequence order
  - $\text{value_avg}(S) < 25; \text{profit_sum}(S) > 160$
  - $\text{max}(S)/\text{avg}(S) < 2; \text{median}(S) - \text{min}(S) > 5$
Timing-Based Constraints in Seq.-Pattern Mining

- **Order constraint**: Some items must happen before the other
  - \{algebra, geometry\} \(\rightarrow\) \{calculus\} (where “\(\rightarrow\)” indicates ordering)
  - Anti-monotonic: Constraint-violating sub-patterns pruned
- **Min-gap/max-gap constraint**: Confines two elements in a pattern
  - E.g., mingap = 1, maxgap = 4
  - Succinct: Enforced directly during pattern growth
- **Max-span constraint**: Maximum allowed time difference between the 1\textsuperscript{st} and the last elements in the pattern
  - E.g., maxspan (S) = 60 (days)
  - Succinct: Enforced directly when the 1\textsuperscript{st} element is determined
- **Window size constraint**: Events in an element do not have to occur at the same time: Enforce max allowed time difference
  - E.g., window-size = 2: Various ways to merge events into elements
Episodes and Episode Pattern Mining

- Episodes and regular expressions: Alternative to seq. patterns
  - Serial episodes: \( A \rightarrow B \)
  - Parallel episodes: \( A \mid B \)
  - Regular expressions: \((A \mid B)C^*(D \rightarrow E)\)

- Ex. Given a large shopping sequence database, one may like to find
  - A, B, C, D, E, such as it follows two constraints
  - Ordering following the template \((A \mid B)C^*(D \rightarrow E)\), and
  - Sum of the prices of A, B, C*, D, and E is greater than $100, where C* means C appears *-times

- How to efficiently mine such sequential patterns?
Summary: Constraint-Based Pattern Mining

- Why Constraint-Based Mining?
- Different Kinds of Constraints: Different Pruning Strategies
- Constrained Mining with Pattern Anti-Monotonicity
- Constrained Mining with Pattern Monotonicity
- Constrained Mining with Data Anti-Monotonicity
- Constrained Mining with Succinct Constraints
- Constrained Mining with Convertible Constraints
- Handling Multiple Constraints
- Constraint-Based Sequential-Pattern Mining
Chapter 7: Advanced Frequent Pattern Mining

- Mining Diverse Patterns
- Sequential Pattern Mining
- Constraint-Based Frequent Pattern Mining
- Graph Pattern Mining
- Pattern Mining Application: Mining Software Copy-and-Paste Bugs
- Summary
What Is Graph Pattern Mining?

- Chem-informatics:
  - Mining frequent chemical compound structures

- Social networks, web communities, tweets, ...
  - Finding frequent research collaboration subgraphs
Frequent (Sub)Graph Patterns

- Given a labeled graph dataset $D = \{G_1, G_2, ..., G_n\}$, the supporting graph set of a subgraph $g$ is $D_g = \{G_i | g \subseteq G_i, G_i \in D\}$

- $\text{support}(g) = \frac{|D_g|}{|D|}$

- A (sub)graph $g$ is **frequent** if $\text{support}(g) \geq \text{min}\_\text{sup}$

- Ex.: Chemical structures

- Alternative:
  - Mining frequent subgraph patterns from a single large graph or network

- Example:
  - $\text{min}\_\text{sup} = 2$
  - Support $= 67\%$

Graph Dataset

Frequent Graph Patterns
Applications of Graph Pattern Mining

- Bioinformatics
  - Gene networks, protein interactions, metabolic pathways
- Chem-informatics: Mining chemical compound structures
- Social networks, web communities, tweets, ...
- Cell phone networks, computer networks, ...
- Web graphs, XML structures, Semantic Web, information networks
- Software engineering: Program execution flow analysis
- Building blocks for graph classification, clustering, compression, comparison, and correlation analysis
- Graph indexing and graph similarity search
Graph Pattern Mining Algorithms: Different Methodologies

- Generation of candidate subgraphs
  - Apriori vs. pattern growth (e.g., FSG vs. gSpan)
- Search order
  - Breadth vs. depth
- Elimination of duplicate subgraphs
  - Passive vs. active (e.g., gSpan [Yan & Han, 2002])
- Support calculation
  - Store embeddings (e.g., GASTON [Nijssen & Kok, 2004], FFSM [Huan, Wang, & Prins, 2003], MoFa [Borgelt & Berthold, ICDM’02])
- Order of pattern discovery
  - Path → tree → graph (e.g., GASTON [Nijssen & Kok, 2004])
Apriori-Based Approach

- The Apriori property (anti-monotonicity): A size-\(k\) subgraph is frequent if and only if all of its subgraphs are frequent.

- A candidate size-(\(k+1\)) edge/vertex subgraph is generated if its corresponding two \(k\)-edge/vertex subgraphs are frequent.

- Iterative mining process:
  - Candidate-generation \(\rightarrow\) candidate pruning \(\rightarrow\) support counting \(\rightarrow\) candidate elimination.
Candidate Generation: Vertex Growing vs. Edge Growing

- Methodology: Breadth-search, Apriori joining two size-$k$ graphs
  - Many possibilities at generating size-$(k+1)$ candidate graphs

- Generating new graphs with one more vertex
  - AGM (Inokuchi, Washio, & Motoda, PKDD’00)

- Generating new graphs with one more edge
  - FSG (Kuramochi & Karypis, ICDM’01)

- Performance shows *via edge growing* is more efficient
Pattern-Growth Approach

- Depth-first growth of subgraphs from \( k \)-edge to \((k+1)\)-edge, then \((k+2)\)-edge subgraphs

- Major challenge
  - Generating many duplicate subgraphs

- Major idea to solve the problem
  - Define an order to generate subgraphs
  - DFS spanning tree: Flatten a graph into a sequence using depth-first search
  - gSpan (Yan & Han, ICDM’02)

\[ G \rightarrow G_1 \rightarrow G_2 \rightarrow \ldots \rightarrow G_n \rightarrow \ldots \]

\[ k\text{-edge} \rightarrow (k+1)\text{-edge} \rightarrow (k+2)\text{-edge} \]

duplicate graphs
**gSPAN: Graph Pattern Growth in Order**

- **Right-most path extension** in subgraph pattern growth
- Right-most path: The path from root to the right-most leaf (choose the vertex with the smallest index at each step)
- Reduce generation of duplicate subgraphs
- **Completeness**: The enumeration of graphs using right-most path extension is **complete**
- DFS code: Flatten a graph into a sequence using depth-first search

<table>
<thead>
<tr>
<th>Edge</th>
<th>Index 1</th>
<th>Index 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_0$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$e_1$</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>$e_2$</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>$e_3$</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>$e_4$</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>$e_5$</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>
Why Mine Closed Graph Patterns?

- Challenge: An $n$-edge frequent graph may have $2^n$ subgraphs
- Motivation: Explore *closed frequent subgraphs* to handle graph pattern explosion problem
- A frequent graph $G$ is *closed* if there exists no supergraph of $G$ that carries the same support as $G$
- *Lossless compression*: Does not contain non-closed graphs, but still ensures that the mining result is complete
- Algorithm CloseGraph: Mines closed graph patterns directly

If this subgraph is *closed* in the graph dataset, it implies that none of its frequent super-graphs carries the same support.
CloseGraph: Directly Mining Closed Graph Patterns

- CloseGraph: Mining closed graph patterns by extending gSpan (Yan & Han, KDD’03)

At what condition can we stop searching their children, i.e., early termination?

- Suppose G and G₁ are frequent, and G is a subgraph of G₁
- If in any part of the graph in the dataset where G occurs, G₁ also occurs, then we need not grow G (except some special, subtle cases), since none of G’s children will be closed except those of G₁
Experiment and Performance Comparison

- The AIDS antiviral screen compound dataset from NCI/NIH
- The dataset contains 43,905 chemical compounds
- Discovered patterns: The smaller minimum support, the bigger and more interesting subgraph patterns discovered

![Chemical structures with minimum support percentages](image)

# of Patterns: Frequent vs. Closed

<table>
<thead>
<tr>
<th>Minimum support</th>
<th>Frequent graphs</th>
<th>Closed frequent graphs</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.05</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.06</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.07</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.08</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Runtime: Frequent vs. Closed

![Runtime graph](image)
Chapter 7: Advanced Frequent Pattern Mining

- Mining Diverse Patterns
- Sequential Pattern Mining
- Constraint-Based Frequent Pattern Mining
- Graph Pattern Mining
- Pattern Mining Application: Mining Software Copy-and-Paste Bugs
- Summary
Pattern Mining Application: Software Bug Detection

- Mining rules from source code
  - Bugs as deviant behavior (e.g., by statistical analysis)
  - Mining programming rules (e.g., by frequent itemset mining)
  - Mining function precedence protocols (e.g., by frequent subsequence mining)
  - Revealing neglected conditions (e.g., by frequent itemset/subgraph mining)
- Mining rules from revision histories
  - By frequent itemset mining
- Mining copy-paste patterns from source code
  - Find copy-paste bugs (e.g., CP-Miner [Li et al., OSDI’04]) (to be discussed here)
Application Example: Mining Copy-and-Paste Bugs

- Copy-pasting is common
  - 12% in Linux file system
  - 19% in X Window system
- Copy-pasted code is error-prone
- Mine “forget-to-change” bugs by sequential pattern mining
  - Build a sequence database from source code
  - Mining sequential patterns
  - Finding mismatched identifier names & bugs

Courtesy of Yuanyuan Zhou@UCSD

```c
void __init prom_meminit(void)
{
    ...... 
    for (i=0; i<n; i++) {
        total[i].adr = list[i].addr; 
        total[i].bytes = list[i].size; 
        total[i].more = &total[i+1]; 
    }
    ...... 

    for (i=0; i<n; i++) {
        taken[i].adr = list[i].addr; 
        taken[i].bytes = list[i].size; 
        taken[i].more = &total[i+1]; 
    }

(Simplified example from linux-2.6.6/arch/sparc/prom/memory.c)
```

Code copy-and-pasted but forget to change “id”!
Building Sequence Database from Source Code

- Statement → number
- Tokenize each component
  - Different operators, constants, key words → different tokens
  - Same type of identifiers → same token
- Program → A long sequence
  - Cut the long sequence by blocks

Map a statement to a number

Tokenize

Hash

<table>
<thead>
<tr>
<th>Hash values</th>
</tr>
</thead>
<tbody>
<tr>
<td>65</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>71</td>
</tr>
</tbody>
</table>

for (i=0; i<n; i++) {
    total[i].adr = list[i].addr;
    total[i].bytes = list[i].size;
    total[i].more = &total[i+1];
}

... 

for (i=0; i<n; i++) {
    taken[i].adr = list[i].addr;
    taken[i].bytes = list[i].size;
    taken[i].more = &total[i+1];
}

Final sequence DB:
(65) (16, 16, 71)
... (65) (16, 16, 71)

Courtesy of Yuanyuan Zhou@UCSD
Sequential Pattern Mining & Detecting “Forget-to-Change” Bugs

- Modification to the sequence pattern mining algorithm
  - Constrain the max gap
- Composing Larger Copy-Pasted Segments
  - Combine the neighboring copy-pasted segments repeatedly
- Find conflicts: Identify names that cannot be mapped to the corresponding ones
  - E.g., 1 out of 4 “total” is unchanged, unchanged ratio = 0.25
  - If \(0 < \text{unchanged ratio} < \text{threshold}\), then report it as a bug
- CP-Miner reported many C-P bugs in Linux, Apache, ... out of millions of LOC (lines of code)

Courtesy of Yuanyuan Zhou@UCSD
Chapter 7: Advanced Frequent Pattern Mining

- Mining Diverse Patterns
- Sequential Pattern Mining
- Constraint-Based Frequent Pattern Mining
- Graph Pattern Mining
- Pattern Mining Application: Mining Software Copy-and-Paste Bugs
- Summary
Summary: Advanced Frequent Pattern Mining

- Mining Diverse Patterns
  - Mining Multiple-Level Associations
  - Mining Multi-Dimensional Associations
  - Mining Quantitative Associations
  - Mining Negative Correlations
  - Mining Compressed and Redundancy-Aware Patterns

- Sequential Pattern Mining
  - Sequential Pattern and Sequential Pattern Mining
  - GSP: Apriori-Based Sequential Pattern Mining
  - SPADE: Sequential Pattern Mining in Vertical Data Format
  - PrefixSpan: Sequential Pattern Mining by Pattern-Growth
  - CloSpan: Mining Closed Sequential Patterns

- Constraint-Based Frequent Pattern Mining
  - Why Constraint-Based Mining?
  - Constrained Mining with Pattern Anti-Monotonicity
  - Constrained Mining with Pattern Monotonicity
  - Constrained Mining with Data Anti-Monotonicity
  - Constrained Mining with Succinct Constraints
  - Constrained Mining withConvertible Constraints
  - Handling Multiple Constraints
  - Constraint-Based Sequential-Pattern Mining

- Graph Pattern Mining
  - Graph Pattern and Graph Pattern Mining
  - Apriori-Based Graph Pattern Mining Methods
  - gSpan: A Pattern-Growth-Based Method
  - CloseGraph: Mining Closed Graph Patterns

- Pattern Mining Application: Mining Software Copy-and-Paste Bugs
References: Mining Diverse Patterns

- R. Srikant and R. Agrawal, “Mining generalized association rules”, VLDB'95
- D. Xin, J. Han, X. Yan and H. Cheng, "On Compressing Frequent Patterns", Knowledge and Data Engineering, 60(1): 5-29, 2007
- D. Xin, H. Cheng, X. Yan, and J. Han, "Extracting Redundancy-Aware Top-K Patterns", KDD'06
- J. Han, H. Cheng, D. Xin, and X. Yan, "Frequent Pattern Mining: Current Status and Future Directions", Data Mining and Knowledge Discovery, 15(1): 55-86, 2007
- F. Zhu, X. Yan, J. Han, P. S. Yu, and H. Cheng, “Mining Colossal Frequent Patterns by Core Pattern Fusion”, ICDE'07
References: Sequential Pattern Mining

- J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu, "Mining Sequential Patterns by Pattern-Growth: The PrefixSpan Approach", IEEE TKDE, 16(10), 2004
- X. Yan, J. Han, and R. Afshar, “CloSpan: Mining Closed Sequential Patterns in Large Datasets”, SDM'03
References: Constraint-Based Frequent Pattern Mining

- R. Ng, L.V.S. Lakshmanan, J. Han & A. Pang, “Exploratory mining and pruning optimizations of constrained association rules”, SIGMOD’98
- G. Grahne, L. Lakshmanan, and X. Wang, “Efficient mining of constrained correlated sets”, ICDE'00
- J. Pei, J. Han, and L. V. S. Lakshmanan, “Mining Frequent Itemsets with Convertible Constraints”, ICDE'01
- J. Pei, J. Han, and W. Wang, “Mining Sequential Patterns with Constraints in Large Databases”, CIKM'02
- F. Bonchi, F. Giannotti, A. Mazzanti, and D. Pedreschi, “ExAnte: Anticipated Data Reduction in Constrained Pattern Mining”, PKDD'03
- F. Zhu, X. Yan, J. Han, and P. S. Yu, “gPrune: A Constraint Pushing Framework for Graph Pattern Mining”, PAKDD'07
References: Graph Pattern Mining

- C. Borgelt and M. R. Berthold, Mining molecular fragments: Finding relevant substructures of molecules, ICDM'02
- J. Huan, W. Wang, and J. Prins. Efficient mining of frequent subgraph in the presence of isomorphism, ICDM'03
- A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data, PKDD'00
- M. Kuramochi and G. Karypis. Frequent subgraph discovery, ICDM'01
- S. Nijssen and J. Kok. A Quickstart in Frequent Structure Mining can Make a Difference. KDD'04
- N. Vanetik, E. Gudes, and S. E. Shimony. Computing frequent graph patterns from semistructured data, ICDM'02
- X. Yan and J. Han, gSpan: Graph-Based Substructure Pattern Mining, ICDM'02
- X. Yan and J. Han, CloseGraph: Mining Closed Frequent Graph Patterns, KDD'03
- X. Yan, P. S. Yu, J. Han, Graph Indexing: A Frequent Structure-based Approach, SIGMOD'04
- X. Yan, P. S. Yu, and J. Han, Substructure Similarity Search in Graph Databases, SIGMOD'05