Chapter 3 Objectives

Understand Sequential Pattern Analysis in the context of transactional data and get a brief introduction to the different algorithms for sequential pattern discovery.

Sequence of Transactions

- Association rule mining searches for relationships between items in a dataset where time is irrelevant.
- Sequential Pattern Analysis considers time (or order of transactions).

Data: sequences of evidences in time order
Target: sub-sequences that happened frequently
Sequence Pattern Examples

- Examples 1
  - 60% of customers typically rent “Star Wars”, then “Empire Strikes Back”, and then “Return of Jedi”.
  - Note: these rentals need not to be consecutive.
  - <SW>,…,<ESB>,…,<RJ>

- Example 2
  - 60% of customers buy “Fitted Sheet and flat sheet and pillow”, followed by “comforter”, followed by “drapes and ruffles”
  - Note: elements of a sequential pattern need not to be simple items.
  - <FittedSheet, FaltSheet, Pillow>, …,<Comforter>,…,<Drapes, Ruffles>

Why sequential pattern mining?

- Time or order in which actions appear or happen can be relevant in decision making.
- Many applications of sequential pattern mining
  - Customer shopping sequences (i.e., for book/video rental):
    - First buy computer, then CD-ROM, and then digital camera, within 3 months.
  - Medical treatment (e.g., symptoms and diseases)
  - Serial crime solving
  - Natural disasters (e.g., earthquakes),
  - Science & engineering processes,
  - Stocks and markets,
  - Telephone calling patterns,
  - Web access log click streams,
  - DNA sequences and gene structures, etc.

Lecture Outline

Part I: Concepts (30 minutes)
  - Basic concepts

Part II: Apriori-based Approaches (45 minutes)
  - Apriori-all
  - GSP

Part III: Pattern-Growth-based Approaches (45 minutes)
  - Free-Span
  - Prefix-Span

Sequence Database

Converts the original transaction database into a database of customer sequences.

Transaction database

- Cust1 (30)
- Cust2 (10,20)
- Cust3 (30, 50, 70)
- Cust4 (30)
- Cust5 (90)
- Cust2 (40, 60, 70)
- Cust4 (90)
- Cust5 (90)

Sequence database

- Cust1 (30)
- Cust2 (10,20)
- Cust3 (30, 50, 70)
- Cust4 (30)
- Cust5 (90)

Sort transactions:
- Customer ID = Major key
- Transaction Time = Minor key

- <a1, a2, a3> not contained in <(10,20)(30)(40,60,70)>
- <(20)(30)(40) contained in <(10,20)(30)(40,60,70)>

CID=1
CID=2
CID=3
CID=4
CID=5

1, <30, 90> 2, <(10,20), 30, (40, 60, 70)> 3, <(30, 50, 70)>
4, <(40, 70), 90> 5, <90>
What Is Sequential Pattern Mining?

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

A sequence database:

<table>
<thead>
<tr>
<th>SID</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>&lt;(ef)(ab)(df)c(b)&gt;</td>
</tr>
<tr>
<td>20</td>
<td>&lt;(ef)(ab)(df)c(b)&gt;</td>
</tr>
<tr>
<td>30</td>
<td>&lt;(ef)(ab)(df)c(b)&gt;</td>
</tr>
<tr>
<td>40</td>
<td>&lt;(ef)(ab)(df)c(b)&gt;</td>
</tr>
</tbody>
</table>

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

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

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

A sequence database:

<table>
<thead>
<tr>
<th>SID</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>&lt;(ef)(ab)(df)c(b)&gt;</td>
</tr>
<tr>
<td>20</td>
<td>&lt;(ef)(ab)(df)c(b)&gt;</td>
</tr>
<tr>
<td>30</td>
<td>&lt;(ef)(ab)(df)c(b)&gt;</td>
</tr>
<tr>
<td>40</td>
<td>&lt;(ef)(ab)(df)c(b)&gt;</td>
</tr>
</tbody>
</table>

Given support threshold min_sup = 2, (ab)c is a sequential pattern (cf. SID 10 & 30)

Sequential Pattern Mining

• Find all the frequent subsequences, i.e. the subsequences whose occurrence frequency in the set of sequences is no less than min_support

Solution – 53 frequent subsequences

Given support threshold min_sup = 2, (ab)c is a sequential pattern (cf. SID 10 & 30)

Subsequence vs. super sequence

• Given two sequences α=<a₁a₂…aₙ> and β=<b₁b₂…bₘ>
  • α is called a subsequence of β, denoted as α⊆β, if there exist integers 1≤j₁<j₂<…<jₙ≤m such that a₁⊆bⱼ₁, a₂⊆bⱼ₂,…, aₙ⊆bⱼₙ
  • β is a super sequence of α
  • A sequence s is maximal if it is not contained in any other sequence.

Sequential Patterns

• Given a set of sequences, where each sequence consists of a list of elements and each element consists of a set of items
  • user-specified min_support threshold

Solution – 53 frequent subsequences

Given support threshold min_sup = 2, (ab)c is a sequential pattern (cf. SID 10 & 30)

Subsequence vs. super sequence

• Given two sequences α=<a₁a₂…aₙ> and β=<b₁b₂…bₘ>
  • α is called a subsequence of β, denoted as α⊆β, if there exist integers 1≤j₁<j₂<…<jₙ≤m such that a₁⊆bⱼ₁, a₂⊆bⱼ₂,…, aₙ⊆bⱼₙ
  • β is a super sequence of α
  • A sequence s is maximal if it is not contained in any other sequence.
Sequence Support Count

- A sequence database is a set of tuples \( <\text{sid}, s> \).
- A tuple \( <\text{sid}, s> \) is said to contain a sequence \( \alpha \), if \( \alpha \) is a subsequence of \( s \), i.e., \( \alpha \subseteq s \).
- The support of a sequence \( \alpha \) is the number of tuples containing \( \alpha \).

<table>
<thead>
<tr>
<th>id</th>
<th>Sequence</th>
<th>( \alpha_1 = &lt;\text{a} &gt; )</th>
<th>support(( \alpha_1 )) = 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>&lt;(abc)(ac)(cf)&gt;</td>
<td>( \alpha_2 = &lt;\text{a} &gt; )</td>
<td>support(( \alpha_2 )) = 4</td>
</tr>
<tr>
<td>20</td>
<td>&lt;(ad)(bc)(ae)&gt;</td>
<td>( \alpha_3 = &lt;\text{ab} &gt; )</td>
<td>support(( \alpha_3 )) = 2</td>
</tr>
<tr>
<td>30</td>
<td>&lt;(ef)(ab)(df)(cb)&gt;</td>
<td>( \alpha_4 = &lt;\text{a} &gt; )</td>
<td>support(( \alpha_4 )) = 4</td>
</tr>
<tr>
<td>40</td>
<td>&lt;(eg)(af)(be)c&gt;</td>
<td>( \alpha_5 = &lt;\text{a} &gt; )</td>
<td>support(( \alpha_5 )) = 4</td>
</tr>
</tbody>
</table>

Counting Sequences (An example)

- A Generated Candidate Pattern:
  - \( \langle 7 \rangle \langle 3,8 \rangle \langle 9 \rangle \langle 4,5,6 \rangle \langle 8 \rangle \)
  - \( \langle 8 \rangle \langle 3,8 \rangle \langle 4,5 \rangle \langle 6 \rangle \langle 7 \rangle \)
  - \( \langle 3 \rangle \langle 4,5 \rangle \langle 8 \rangle \)

- IF Min_Sup \( \leq 50\% \) THEN \( \langle 3 \rangle \langle 4,5 \rangle \langle 8 \rangle \) is Frequent

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.
- Comparison of association rules and sequence mining:
  - Mining for association rules:
    - Purpose: Discovery of frequent unordered itemsets.
    - Complexity: With \( n \) items there are \( \binom{n}{k} \) \( k \)-itemsets (sets with \( k \) items).
  - Mining for Sequential Patterns:
    - Purpose: Discovery of frequent sequences of (unordered) itemsets.
    - Complexity: With \( n \) items there are \( n^k \) sequences (sequences with \( k \) items).

Association mining algorithms discover isolated item sets (intra-event patterns). Sequence mining algorithms discover series of item sets (inter-event patterns).

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):
- Data Projection-based approaches:
  - FreeSpan (Han et al. “frequent pattern-projected sequential pattern mining” SIGKDD 2000).
- Mining sequential patterns with constraints:
Lecture Outline

Part I: Concepts (30 minutes)
• Basic concepts

Part II: Apriori-based Approaches (45 minutes)
• Apriori-All
• GSP

Part III: Pattern-Growth-based Approaches (45 minutes)
• Free-Span
• Prefix-Span

AprioriAll: The idea

• Basic method to mine sequential patterns
  – Based on the Apriori algorithm
  – Count all the large sequences, including non-maximal sequences
  – Use Apriori-generate function to generate candidate sequences: get candidates for a pass using only the large sequences found in the previous pass and make a scan over the data to find their support

AprioriAll Algorithm(1)

• AprioriAll Algorithm
  \[ C_k : \text{Candidate sequence of size } k \]
  \[ L_k : \text{frequent or large sequence of size } k \]

\[ L_1 = \{\text{large 1-sequence}\}; \quad \text{//result of itemset phase} \]

\[ \text{for } (k = 2; L_{k-1} \neq \emptyset; k++) \text{ do begin} \]
  \[ C_k = \text{candidates generated from } L_{k-1}; \]
  \[ \text{for each customer-sequence } c \text{ in database do} \]
  \[ \text{Increment the count of all candidates in } C_k \text{ that are contained in } c \]
  \[ L_k = \text{Candidates in } C_k \text{ with minimum support} \]
\[ \text{end} \]

\[ \text{Answer=Maximal sequences in } \bigcup \text{ } L_k; \]

• Candidate Generation --Join Step:
  \[ C_k \text{ is generated by joining } L_{k-1} \text{ with itself} \]
  \[ \text{Insert into } C_k; \]
  \[ \text{Select } p.litemset_1, \ldots, p.litemset_{k-1}, q.litemset_{k-1} \]
  \[ \text{From } L_{k-1}, p, q \]
  \[ \text{Where } p.litemset_1 = q.litemset_1, \ldots, \]
  \[ p.litemset_{k-2} = q.litemset_{k-2} \]

For example:
\[ \{1,2,3\} \times \{1,2,4\} = \{1,2,3,4\} \]
and
\[ \{1,2,4,3\} \]

AprioriAll: The big picture

Five-phase algorithm
1. Sort phase:
   Create the sequence database from transactions.
2. Large itemset phase
   Find all frequent itemsets using Apriori
3. Transformation phase:
   Do integer mapping for large itemsets
4. Sequence phase:
   Find all frequent sequential patterns using Apriori.
5. Maximal phase:
   Eliminate non maximal sequences.
Sequence Database Example

<table>
<thead>
<tr>
<th>Customer ID</th>
<th>TransactionTime</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>90</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>10,20</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>40,60,70</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>30,50,70</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>40,70</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>90</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>90</td>
</tr>
</tbody>
</table>

MinSupport = 40%, i.e. 2 customers

Answer: (<30><90>) (CID1,4) (<30><40,70>) (CID2,4)

Not Answer: <30> <40><70><90> (<30><40>) (<30><70>) (<40><70>)

Why?

Sort Phases

- Sort Phases
  CID: major key, TID: secondary key

<table>
<thead>
<tr>
<th>Customer ID</th>
<th>TransactionTime</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>90</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>10,20</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>40,60,70</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>30,50,70</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>40,70</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>90</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>90</td>
</tr>
</tbody>
</table>

Transformation Phase

- Transformation Phase:
  Each large itemset is then mapped to a set of contiguous integers
  (Why? to be able to compare two frequent itemsets in constant time)
  - Represent transactions as sets of large itemsets.

Litemset Phase

- Litemset Phase:
  - Find all large itemset
    (Why? Because each itemset in a large sequence has to be a large itemset.)
  - To get large (frequent) itemsets → Use Apriori algorithm
  - Need to modify support counting. (For sequential patterns, support is measured by fraction of customers.)

<table>
<thead>
<tr>
<th>Customer ID</th>
<th>TransactionTime</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>90</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>10,20</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>30</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>40,60,70</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>30,50,70</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>40,70</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>90</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>90</td>
</tr>
</tbody>
</table>

MinSupport = 40%, i.e. 2 customers

Litemset Result:
{30} {40} {70} {40 70} {90}

Difference from Apriori:
- the support count should be incremented only once per customer
Sequence Phase

- **Sequence Phase:**
  - Use set of large itemsets to find the desired sequences.
  - Similar structure to Apriori algorithms used to find large itemsets.
    - Use seed set to generate candidate sequences.
    - Count support for each candidate.
    - Eliminate candidate sequences which are not large.

Maximal phase

- **Maximum Phase:**
  - Find the maximal sequences among the set of frequent sequences.
  - Delete all sequences that are sub-sequences of other frequent sequences.

```
for (k=n; k>1; k--) do
  for each k-sequence S_k do
    Delete from all subsequences of S_k
```

Summary for AprioriAll

- Algorithm wastes much time in counting non-maximal sequences, which can not be sequential patterns.
- There are other variations of AprioriAll that reduce the candidates that are not maximal: AprioriSome and DynamicSome.
- Absence of time window constraints.
- AprioriALL is the basis of many efficient algorithm developed later. GSP is among them.

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
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>

<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)bfabf&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>

Given support threshold min_sup = 2

<table>
<thead>
<tr>
<th>Sequence</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>2</td>
</tr>
<tr>
<td>&lt;f&gt;</td>
<td>1</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Seq. ID</th>
<th>Sequence</th>
<th>Cand</th>
<th>Sup</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>&lt;(bd)cb(ac)&gt;</td>
<td>&lt;a&gt;</td>
<td>3</td>
</tr>
<tr>
<td>20</td>
<td>&lt;(bf)ce)b(fg)&gt;</td>
<td>&lt;b&gt;</td>
<td>5</td>
</tr>
<tr>
<td>30</td>
<td>&lt;(ah)bfabf&gt;</td>
<td>&lt;c&gt;</td>
<td>4</td>
</tr>
<tr>
<td>40</td>
<td>&lt;(be)ce)d&gt;</td>
<td>&lt;d&gt;</td>
<td>3</td>
</tr>
<tr>
<td>50</td>
<td>&lt;a(bd)bcb(ade)&gt;</td>
<td>&lt;e&gt;</td>
<td>2</td>
</tr>
</tbody>
</table>

Generating Length-2 Candidates

51 length-2 Candidates

Without Apriori property, 8*8+8*7/2=92 candidates
Apriori prunes 44.57% candidates

The GSP Algorithm

- Take sequences in form of <x> as length-1 candidates
- Scan database once, find F_1, the set of length-1 sequential patterns
- Let k=1; while F_k is not empty do
  - Form C_{k+1}, the set of length-(k+1) candidates from F_k;
  - If C_{k+1} is not empty, scan database once, find F_{k+1}, the set of length-(k+1) sequential patterns
  - Let k=k+1;
Generating Length-2 Candidates

<table>
<thead>
<tr>
<th>Seq. ID</th>
<th>Sequence</th>
<th>&lt;a&gt;</th>
<th>&lt;b&gt;</th>
<th>&lt;c&gt;</th>
<th>&lt;d&gt;</th>
<th>&lt;e&gt;</th>
<th>&lt;f&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>(bd)cb(ac)</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>(bf)ce(bf)fg</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>(ah)(bf)abf</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>40</td>
<td>(be)(ce)d</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>50</td>
<td>(a(bd)bcb(ade)</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

SID: 30, 50

SID: 20, 30

Length-2 Sequential Patterns

- After scanning the database to collect support count for each length-2 candidate
- There are 19 length-2 candidates which pass the minimum support threshold
  - They are length-2 sequential patterns
    - 16 of them in the pattern of <xy>
    - 3 of them in the pattern of <(xy)>

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 → <aba> is a length-3 candidate
      - 54 candidates are generated
    - <bd>, <bb> and <db> are all length-2 sequential patterns → <(bd)b> is a length-3 candidate
      - 27 candidates are generated

- Find Length-3 Sequential Patterns
  - Scan database once more, collect support counts for candidates
  - 19 out of 81 candidates pass support threshold
Generating Length-3 Candidates

Example of generating \(<xyz>\) pattern for \(<ag>:\n• Need to concatenate another Length-2 frequent itemset
• Concatenating another frequent itemsets that start with \(a\) to form \(<aaa>\) and \(<aab>\):

<table>
<thead>
<tr>
<th>Seq. ID</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>&lt;(bd)c(bac)&gt;</td>
</tr>
<tr>
<td>20</td>
<td>&lt;(bf)(ce)(bg)&gt;</td>
</tr>
<tr>
<td>30</td>
<td>&lt;(ab)(bf)(ab)&gt;</td>
</tr>
<tr>
<td>40</td>
<td>&lt;(be)(ce)&gt;</td>
</tr>
<tr>
<td>50</td>
<td>&lt;(ab)(df)(bade)&gt;</td>
</tr>
</tbody>
</table>

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

Example of generating \(<xy)z\) pattern for \(<bd>:\n• Need to concatenate another Length-2 frequent itemset
• Concatenating those patterns that start with \(b\) or \(d\) to form something like \(<a(bd)>, <b(bd), <c(bd), <d(bd), <f(bd)>
• Concatenating those patterns that starts with \(b\) or \(d\) to form something like \((bd)a>, (bd)b>, (bd)c>, (bd)d>, (bd)e>, (bd)f>

The GSP Mining Process

<table>
<thead>
<tr>
<th>Seq. ID</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>(&lt;(bd)c(bac)&gt;)</td>
</tr>
<tr>
<td>20</td>
<td>(&lt;(bf)(ce)(bg)&gt;)</td>
</tr>
<tr>
<td>30</td>
<td>(&lt;(ab)(bf)(ab)&gt;)</td>
</tr>
<tr>
<td>40</td>
<td>(&lt;(be)(ce)&gt;)</td>
</tr>
<tr>
<td>50</td>
<td>(&lt;(ab)(df)(bade)&gt;)</td>
</tr>
</tbody>
</table>

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

Bottlenecks of GSP

• A huge set of candidates could be generated
  – 1,000 frequent length-1 sequences generate \(1,000 \times 1,000 = 1,000,000\) length-2 candidates!

  \[1000 \times 1000 + \frac{1000 \times 999}{2} = 1,499,500\]

• Multiple scans of database
• Real challenge: mining long sequential patterns
  – An exponential number of short candidates
  – A length-100 sequential pattern needs \(10^{30}\) candidate sequences!

\[
\sum_{i=1}^{100} \binom{100}{i} = 2^{100} - 1 \approx 10^{30}
\]
Lecture Outline

Part I: Concepts  (30 minutes)
  • Basic concepts

Part II: Apriori-based Approaches  (45 minutes)
  • Apriori-All
  • GSP

Part III: Pattern-Growth-based Approaches  (45 minutes)
  • Free-Span (Frequent Pattern-Projected Sequential Pattern Mining)
  • Prefix-Span (Prefix-Projected Sequential Pattern)

FreeSpan (Generalities)

• J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M.-C. Hsu. FreeSpan: Frequent pattern-projected sequential pattern mining. KDD'00, pages 355-359.

• A divide-and-conquer approach
  - Recursively project a sequence database into a set of smaller databases
  - Mine each projected database to find the subset of patterns

FreeSpan (example)

• Given a sequence database S and min_support = 2
  • Step 1: find length-1 sequential patterns and list them in support descending order
    - \textit{f} \_list = a:4, b:4, c:4, d:3, e:3, f:3; g:1
  • Step 2: divide search space. The complete set of seq. patterns can be partitioned into 6 disjoint subsets (move down the \textit{f} \_list):
    - ones only contain item a
    - ones contain item b but no items after b in \textit{f} \_list
    - ones contain item c but no items after c in \textit{f} \_list
    - ones contain item d but no items after d in \textit{f} \_list
    - ones contain item e but no items after e in \textit{f} \_list
    - ones contain item f

find subsets of sequential patterns. They can be mined by constructing projected databases and mining each recursively

\begin{tabular}{|c|c|}
\hline
\textbf{SID} & \textbf{Sequence} \\
\hline
10 & \langle a \langle ab \rangle c \langle ac \rangle d \langle cf \rangle \rangle \rangle \\
20 & \langle a \langle ad \rangle c \langle bc \rangle \langle ae \rangle \rangle \rangle \\
30 & \langle a \langle ef \rangle \langle ab \rangle \langle df \rangle d \rangle \rangle \rangle \\
40 & \langle a \langle eg \rangle \langle af \rangle \rangle \rangle \rangle \\
\hline
\end{tabular}
From FreeSpan to PrefixSpan

- **FreeSpan**:
  - Projection-based: No candidate sequence needs to be generated.
  - But, projection can be performed at any point in the sequence, and the projected sequences may not shrink much. For example, the size of f-projected database is the same as the original sequence database.

- **PrefixSpan**:
  - Projection-based.
  - But only prefix-based projection: less projections and quickly shrinking sequences.

---

Prefix of a Sequence

- Given two sequences \( \alpha = a_1 a_2 \ldots a_n \) and \( \beta = b_1 b_2 \ldots b_m \), \( m \leq n \).
- Sequence \( \beta \) is called a prefix of \( \alpha \) if and only if:
  - \( b_i = a_i \) for \( i \leq m-1 \);
  - \( b_m \subseteq a_m \);
  - All the items in \( (a_m - b_m) \) are alphabetically after those in \( b_m \).

**Given an alphabetical order of items in each itemset (element)**

- \( \alpha = <a(b)c)(ac)d(cf)> \)
- \( \beta = <a, c) > \)
- \( \gamma = <(c)d(cf)> \)

---

Projection

- Given sequences \( \alpha \) and \( \beta \), such that \( \beta \) is a subsequence of \( \alpha \).
- A subsequence \( \alpha' \) of sequence \( \alpha \) is called a projection of \( \alpha \) w.r.t. \( \beta \) prefix if and only if:
  - \( \alpha' \) has prefix \( \beta \);
  - There exist no proper super-sequence \( \alpha'' \) of \( \alpha' \) such that \( \alpha'' \) is a subsequence of \( \alpha \) and also has prefix \( \beta \).

- \( \alpha = <a(b)c)(ac)d(cf)> \)
- \( \beta = <(bc)a> \)
- \( \alpha' = <(bc)(ac)d(cf)> \)

---

Postfix

- Let \( \alpha' = <a_1 a_2 \ldots a_n a_m> \) be the projection of \( \alpha \) w.r.t. prefix \( \beta = <a_1 a_2 \ldots a_{m-1} a'_m> \) (\( m \leq n \)).
- Sequence \( \gamma = <a''_m a_{m+1} \ldots a_n> \) is called the postfix of \( \alpha \) w.r.t. prefix \( \beta \), denoted as \( \gamma = \alpha/\beta \), where \( a''_m = (a_m - a'_m) \).
- We also denote \( \alpha = \beta \cdot \gamma \).

- \( \alpha' = <a(b)c)(ac)d(cf)> \)
- \( \beta = <a(b)c) > \)
- \( \gamma = <(c)d(cf)> \)
PrefixSpan – Algorithm

- **Input**: A sequence database $S$, and the minimum support threshold $\text{min}_\text{sup}$
- **Output**: The complete set of sequential patterns
- **Method**: Call PrefixSpan(\(<\),0,$S$)
- **Subroutine** PrefixSpan($\alpha$, l, $S|_\alpha$)

**Parameters**:
- $\alpha$: sequential pattern,
- l: the length of $\alpha$;
- $S|_\alpha$: the $\alpha$-projected database, if $\alpha \neq \langle \rangle$; otherwise, the sequence database $S$.

PrefixSpan – Algorithm (2)

- **Method**
  1. Scan $S|_\alpha$ once, find the set of frequent items $b$ such that:
     a) $b$ can be assembled to the last element of $\alpha$ to form a sequential pattern; or
     b) $\langle b \rangle$ can be appended to $\alpha$ to form a sequential pattern.
  2. For each frequent item $b$, append it to $\alpha$ to form a sequential pattern $\alpha'$, and output $\alpha'$;
  3. For each $\alpha'$, construct $\alpha'$-projected database $S|_{\alpha'}$, and call PrefixSpan($\alpha'$, l+1, $S|_{\alpha'}$).

PrefixSpan - Example

1. Find length-1 sequential patterns

<table>
<thead>
<tr>
<th>id</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>&lt;(abc)(ac)d(cf)&gt;</td>
</tr>
<tr>
<td>20</td>
<td>&lt;(cd)(bc)(ae)&gt;</td>
</tr>
<tr>
<td>30</td>
<td>&lt;(ef)(ab)(dch)&gt;</td>
</tr>
<tr>
<td>40</td>
<td>&lt;(c)(af)ehc&gt;</td>
</tr>
</tbody>
</table>

   min_support = 2

2. Divide search space

   - $<$a$>$
   - $<$b$>$
   - $<$c$>$
   - $<$d$>$
   - $<$e$>$
   - $<$f$>$

   Partition search space into 6 subsets: ones having prefix $<$a$>$; ones having prefix $<$b$>$; ... ones having prefix $<$f$>$;

Let’s see the case of $<$d$>$

PrefixSpan – Example (2)

3. Find subsets of sequential database for $<$d$>$

   <db> <dc>

Projected database for $<$d$>$
PrefixSpan - characteristics

- No candidate sequence needs to be generated by PrefixSpan
- Projected databases keep shrinking
- The major cost of PrefixSpan is the construction of projected databases

**How to reduce this cost?**

Different projection methods

- **Bi-level projection**
  - reduces the number and the size of projected databases
- **Pseudo-Projection**
  - reduces the cost of projection when projected database can be held in main memory

Bi-level Projection

- Scan to get 1-length sequences
- Construct a **triangular matrix** instead of projected databases for each length-1 patterns

**Bi-level projection (2)**

- For each length-2 sequential pattern \( \alpha \), construct the \( \alpha \)-projected database and find the frequent items
- Construct corresponding S-matrix

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>(&lt;a(bac)(ac)(cf)&gt;&lt;c&gt;)</td>
<td>4</td>
</tr>
<tr>
<td>(&lt;_c)a&gt;)</td>
<td>2</td>
</tr>
<tr>
<td>(&lt;c&gt;)</td>
<td>1</td>
</tr>
</tbody>
</table>

**Bi-level projection (3) - optimization**

- “Do we need to include every item in a postfix in the projected databases?”
- NO! Item pruning in projected database by 3-way Apriori checking

- **Support** \(<a(bd)>\) = 2
- **Support** \(<a(bc)>\) = 1
- **Support** \(<a(c)>\) = 3
Pseudo-Projection

- **Observation**: postfixes of a sequence often appear repeatedly in recursive projected databases
- **Method**: instead of constructing *physical* projection by collecting all the postfixes, we can use pointers referring to the sequences in the database as a pseudo-projection
- Every projection consists of two pieces of information: pointer to the sequence in database and offset to the postfix in the sequence

<table>
<thead>
<tr>
<th>Pointer</th>
<th>Offset</th>
<th>Postfix</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>2</td>
<td>&lt;(abc)(ac)d(cf)&gt;</td>
</tr>
<tr>
<td>s1</td>
<td>5</td>
<td>&lt;(ac)d(cf)&gt;</td>
</tr>
<tr>
<td>s1</td>
<td>6</td>
<td>&lt;(_c)d(cf)&gt;</td>
</tr>
</tbody>
</table>

Summary

- Sequential Pattern Mining is useful in many application, e.g. weblog analysis, financial market prediction, BioInformatics, etc.

- It is similar to the frequent itemsets mining, *but* with consideration of ordering.

- We have looked at different approaches that are descendants from two popular algorithms in mining frequent itemsets
  - Candidates Generation: AprioriAll and GSP
  - Pattern Growth: FreeSpan and PrefixSpan