RULEGROWTH:
Mining Sequential Rules Common to Several Sequences by Pattern-Growth

Philippe Fournier-Viger¹
Roger Nkambou²
Vincent Shin-Mu Tseng¹

¹National Cheng Kung University (Taiwan)
²University of Québec in Montréal (Canada)
Introduction

• Many **databases** contain large amount of **temporal or sequential information**.

• It is a challenge to develop algorithms for discovering useful patterns in these databases.

• Different kind of **temporal** patterns: repetitive patterns, trends, similar patterns, sequential patterns, etc.

• In this paper, we are interested by **sequence databases** containing sequences of **discrete events or symbols**.
Sequences of webpage clicks

Webpage A → Webpage B → Webpage C → ...

Sequences of activities

Home → Watching movies → Visit museum → ...
Sequences of purchases

Computer ➔ Monitor ➔ Router ➔ ...

Sequences of words

Where ➔ are ➔ you ➔ going?
Sequence Database

- Let there be a set of symbols (e.g. $a, b, c, d\ldots g$) called **items**.
- An **itemset** is a set of **items** that appeared simultaneously.
- Each **sequence** is an ordered list of itemsets.

<table>
<thead>
<tr>
<th>ID</th>
<th>Sequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq1</td>
<td>{a, b}, {c}, {f}, {g}, {e}</td>
</tr>
<tr>
<td>seq2</td>
<td>{a, d}, {c}, {b}, {a, b, e, f}</td>
</tr>
<tr>
<td>seq3</td>
<td>{a}, {b}, {f}, {e}</td>
</tr>
<tr>
<td>seq4</td>
<td>{b}, {f, g}</td>
</tr>
</tbody>
</table>
Sequential pattern mining

Input:
- A sequence database (a set of sequences)
- A \textit{minsup} threshold

Output:
- All sub-sequences having a support greater or equal to \textit{minsup}.

Example: \textit{minsup} = 50 \% (2 sequences)

<table>
<thead>
<tr>
<th>IFD</th>
<th>Sequence</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{a}, {a,b,c} {a, c} {d} {c, f}</td>
<td>100 %</td>
</tr>
<tr>
<td>2</td>
<td>{a, d}, {c} {b, c} {a, e}</td>
<td>50 %</td>
</tr>
<tr>
<td>3</td>
<td>{e, f}, {a, b} {d, f} {c} {b}</td>
<td>50 %</td>
</tr>
<tr>
<td>4</td>
<td>{e}, {g}, {a, f} {c} {b} {c}</td>
<td>...</td>
</tr>
</tbody>
</table>

Sequential patterns

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>{a}</td>
<td>100 %</td>
</tr>
<tr>
<td>{a}, {b,c}</td>
<td>50 %</td>
</tr>
<tr>
<td>{a, b}</td>
<td>50 %</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Limitation of sequential pattern mining

• SPM is not very useful for making predictions.
• For example, consider the pattern \{x\},\{y\}.

<table>
<thead>
<tr>
<th>IFD</th>
<th>sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>{x}, {w}, {z}, {y}</td>
</tr>
<tr>
<td>2</td>
<td>{x}, {z}, {z}</td>
</tr>
<tr>
<td>3</td>
<td>{x}, {z}, {y}</td>
</tr>
<tr>
<td>4</td>
<td>{x}, {z}, {z}</td>
</tr>
</tbody>
</table>

• Although \(y\) appears frequently after \(x\), there are also many cases where \(x\) is not followed by \(y\).
• If we want to make predictions, we need a measurement of the confidence that if \(x\) occurs, it will be followed by \(y\).
A solution: sequential Rule Mining

• **Sequential rules**: a type of sequential patterns that incorporate a measure of **confidence**.

• A **sequential rule** typically has the form \( X \rightarrow Y \) and has a **confidence** and a **support**.
  
  \( \{bread, milk\} \rightarrow \{coffee\} \)  confidence : 75 %

• **Several algorithms** by Manila et al. (1997), Hamilton & Karimi (2005), Hsieh (2006), Deogun (2005). But mostly for discovering rules in a **single sequence**.

• In this paper, we are interested by finding rules appearing in **multiple sequences**.
Finding sequential rule in multiple sequences?

• Zaki et al. (2001) proposed the RuleGen algorithm
• Rules of the form $X \rightarrow Y$ where $X$ and $Y$ are sequential patterns.

A sequence database

1: \{Vivaldi\}, \{Mozart\}, \{Handel\}, \{Berlioz\}
2: \{Mozart\}, \{Bach\}, \{Paganini\}, \{Vivaldi\}, \{Handel\}, \{Berlioz\}
3: \{Handel\}, \{Vivaldi\}, \{Mozart\}, \{Ravel\}, \{Berlioz\}
4: \{Vivaldi\}, \{Mozart\}, \{Handel\}, \{Bach\}, \{Berlioz\}
5: \{Mozart\}, \{Bach\}, \{Vivaldi\}, \{Handel\}
6: \{Vivaldi\}, \{Handel\}, \{Mozart\}, \{Bach\}

Some sequential rules

R1: \{Vivaldi\}, \{Mozart\}, \{Handel\} $\Rightarrow$ \{Berlioz\}
R2: \{Mozart\}, \{Vivaldi\}, \{Handel\} $\Rightarrow$ \{Berlioz\},
R3: \{Handel\}, \{Vivaldi\}, \{Mozart\} $\Rightarrow$ \{Berlioz\},
R4: \{Handel, Vivaldi\}, \{Mozart\} $\Rightarrow$ \{Berlioz\},
R5: \{Handel\}, \{Vivaldi, Mozart\} $\Rightarrow$ \{Berlioz\},
R6: \{Handel, Vivaldi, Mozart\} $\Rightarrow$ \{Berlioz\}.

Finding sequential rule in multiple sequences?

• Zaki et al. (2001) proposed the **RuleGen** algorithm
• Rules of the form \( X \rightarrow Y \) where \( X \) and \( Y \) are **sequential patterns**.

**A sequence database**

1: \{Vivaldi\}, \{Mozart\}, \{Handel\}, \{Berlioz\}
2: \{Mozart\}, \{Bach\}, \{Paganini\}, \{Vivaldi\}, \{Handel\}, \{Berlioz\}
3: \{Handel\}, \{Vivaldi\}, \{Mozart\}, \{Ravel\}, \{Berlioz\}
4: \{Vivaldi\}, \{Mozart\}, \{Handel\}, \{Bach\}, \{Berlioz\}
5: \{Mozart\}, \{Bach\}, \{Vivaldi\}, \{Handel\}
6: \{Vivaldi\}, \{Handel\}, \{Mozart\}, \{Bach\}

**Some sequential rules**

<table>
<thead>
<tr>
<th>Rule</th>
<th>Left Side</th>
<th>Right Side</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>{Vivaldi}, {Mozart}, {Handel}</td>
<td>{Berlioz}</td>
</tr>
<tr>
<td>R2</td>
<td>{Mozart}, {Vivaldi}, {Handel}</td>
<td>{Berlioz}</td>
</tr>
<tr>
<td>R3</td>
<td>{Handel}, {Vivaldi}, {Mozart}</td>
<td>{Berlioz}</td>
</tr>
<tr>
<td>R4</td>
<td>{Handel}, {Vivaldi}, {Mozart}</td>
<td>{Berlioz}</td>
</tr>
<tr>
<td>R5</td>
<td>{Handel}, {Vivaldi}, {Mozart}</td>
<td>{Berlioz}</td>
</tr>
<tr>
<td>R6</td>
<td>{Handel}, {Vivaldi}, {Mozart}</td>
<td>{Berlioz}</td>
</tr>
</tbody>
</table>

**Problem**: all these rules are very similar!!!... There are 23 such rules with these items.

- **R1** support: 33% confidence: 100%
- **R2** support: 16%, confidence: 50%
- **R3** support: 16%, confidence: 100%
Our solution: partially-ordered sequential rules

Some sequential rules

R1: \{Vivaldi\}, \{Mozart\}, \{Handel\} ⇒ \{Berlioz\}
R2: \{Mozart\}, \{Vivaldi\}, \{Handel\} ⇒ \{Berlioz\},
R3: \{Handel\}, \{Vivaldi\}, \{Mozart\} ⇒ \{Berlioz\},
R4: \{Handel, Vivaldi\}, \{Mozart\} ⇒ \{Berlioz\},
R5: \{Handel\}, \{Vivaldi, Mozart\} ⇒ \{Berlioz\},
R6: \{Handel, Vivaldi, Mozart\} ⇒ \{Berlioz\}.

By removing the order on the left side or right side of a rule, we can obtain a single rule:

\{Mozart, Vivaldi, Handel\} ⇒ \{Berlioz\}

support: 75%  confidence: 66%

Our solution: partially-ordered sequential rules

• A **sequential rule** $X \Rightarrow Y$ is a relationship between two disjoint and non empty itemsets $X,Y$.

• A sequential rule $X \Rightarrow Y$ has **two properties**:
  – **Support**: the number of sequences where $X$ occurs before $Y$, divided by the number of sequences.
  – **Confidence** the number of sequences where $X$ occurs before $Y$, divided by the number of sequences where $X$ occurs.

• **The task**: finding all **valid rules**, rules with a support and confidence not less than user-defined thresholds $\text{minSup}$ and $\text{minConf}$ (Fournier-Viger, 2010).
An example of Sequential Rule Mining

Consider $\text{minSup}= 0.5$ and $\text{minConf}= 0.5$:

A sequence database

<table>
<thead>
<tr>
<th>ID</th>
<th>Sequences</th>
<th>Support</th>
<th>Confidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq1</td>
<td>{a, b}, {c}, {f}, {g}, {e}</td>
<td>0.5</td>
<td>1.0</td>
</tr>
<tr>
<td>seq2</td>
<td>{a, d}, {c}, {b}, {a, b, e, f}</td>
<td>0.5</td>
<td>0.66</td>
</tr>
<tr>
<td>seq3</td>
<td>{a}, {b}, {f}, {e}</td>
<td>0.5</td>
<td>1.0</td>
</tr>
<tr>
<td>seq4</td>
<td>{b}, {f, g}</td>
<td>0.75</td>
<td>0.75</td>
</tr>
</tbody>
</table>

Some rules found

<table>
<thead>
<tr>
<th>ID</th>
<th>Rule</th>
<th>Support</th>
<th>Confidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>r1</td>
<td>{a, b, c} $\Rightarrow$ {e}</td>
<td>0.5</td>
<td>1.0</td>
</tr>
<tr>
<td>r2</td>
<td>{a} $\Rightarrow$ {c, e,f}</td>
<td>0.75</td>
<td>0.75</td>
</tr>
<tr>
<td>r3</td>
<td>{a, b} $\Rightarrow$ {e, f}</td>
<td>0.75</td>
<td>1.0</td>
</tr>
<tr>
<td>r4</td>
<td>{b} $\Rightarrow$ {e, f}</td>
<td>0.75</td>
<td>1.0</td>
</tr>
<tr>
<td>r5</td>
<td>{a} $\Rightarrow$ {e, f}</td>
<td>0.5</td>
<td>1.0</td>
</tr>
<tr>
<td>r6</td>
<td>{c} $\Rightarrow$ {f}</td>
<td>0.5</td>
<td>1.0</td>
</tr>
<tr>
<td>r7</td>
<td>{a} $\Rightarrow$ {b}</td>
<td>0.5</td>
<td>0.66</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Previous Algorithms

• **CMRules (2010)**: An association rule mining based algorithm for the discovery of sequential rules.

• **CMDeo (2010)**: An Apriori based algorithm for the discovery of sequential rules.

• **Limitation**: Both algorithms use a « generate-candidate-and-test » approach that may generate a large amount of candidates for dense datasets. Many candidates do not appear in the database.
RuleGrowth

An algorithm inspired by PrefixSpan. It generates rules by growing them one item at a time.

The input is:

- A sequence database:

<table>
<thead>
<tr>
<th>ID</th>
<th>Sequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq1</td>
<td>{a, b}, {c}, {f}, {g}, {e}</td>
</tr>
<tr>
<td>seq2</td>
<td>{a, d}, {c}, {b}, {a, b, e, f}</td>
</tr>
<tr>
<td>seq3</td>
<td>{a}, {b}, {f}, {e}</td>
</tr>
<tr>
<td>seq4</td>
<td>{b}, {f, g}</td>
</tr>
</tbody>
</table>

- minsup = 0.5 %,
- minconf = 0.5%
**RuleGrowth**

**Step 1:** Scan database to calculate the support of each item. Keep only frequent items.

<table>
<thead>
<tr>
<th>ID</th>
<th>Sequences</th>
<th>Frequent items:</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq1</td>
<td>{a, b}, {c}, {f}, {g}, {e}</td>
<td>Item</td>
</tr>
<tr>
<td>seq2</td>
<td>{a, d}, {c}, {b}, {a, b, e, f}</td>
<td>a</td>
</tr>
<tr>
<td>seq3</td>
<td>{a}, {b}, {f}, {e}</td>
<td>b</td>
</tr>
<tr>
<td>seq4</td>
<td>{b}, {f, g}</td>
<td>c</td>
</tr>
<tr>
<td></td>
<td></td>
<td>d</td>
</tr>
<tr>
<td></td>
<td></td>
<td>e</td>
</tr>
<tr>
<td></td>
<td></td>
<td>f</td>
</tr>
</tbody>
</table>
RuleGrowth

Step 2: For each pairs of frequent items, try to create a rule with only two items. e.g. \{a\}⇒\{b\}.

<table>
<thead>
<tr>
<th>ID</th>
<th>Sequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq1</td>
<td>{a, b}, {c}, {f}, {g}, {e}</td>
</tr>
<tr>
<td>seq2</td>
<td>{a, d}, {c}, {b}, {a, b, e, f}</td>
</tr>
<tr>
<td>seq3</td>
<td>{a}, {b}, {f}, {e}</td>
</tr>
<tr>
<td>seq4</td>
<td>{b}, {f, g}</td>
</tr>
</tbody>
</table>

Frequent items:

<table>
<thead>
<tr>
<th>Item</th>
<th>support</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>75 %</td>
</tr>
<tr>
<td>b</td>
<td>100 %</td>
</tr>
<tr>
<td>c</td>
<td>50 %</td>
</tr>
<tr>
<td>d</td>
<td>25 %</td>
</tr>
<tr>
<td>e</td>
<td>75 %</td>
</tr>
<tr>
<td>f</td>
<td>100 %</td>
</tr>
</tbody>
</table>

For each rule, calculate the confidence and support. If the confidence and support are respectively higher or equal to minconf and minsup, the rule is output.

\{a\}⇒\{b\}. Support = 50 % Confidence = $\frac{2}{3}$
RuleGrowth

**Step3:** Find larger rules by recursively scanning the database for adding a single item at a time to the left or right part of each rule (these processes are called *left* and *right expansions*).

- `{a}⇒{b, e, f}`
- `{a}⇒{b, f}`
- `{a}⇒{b, e}`
- `{a}⇒{c}`
- `{a}⇒{b}`
- `{a, c}⇒{b}`

For each rule, calculate the confidence and support. If the confidence and support are respectively higher or equal to minconf and minsup, output the rule.
RuleGrowth

• When a rule should be expanded?
  A rule should be expanded only if it has the minimum support.
RuleGrowth

How to choose items for performing left expansions of a rule $X \Rightarrow Y$?

Scan the sequences containing the rule and note items appearing in at least $\text{minsup} \times |S|$ sequences before the last occurrence of $Y$.

For example:  $\{a\} \Rightarrow \{b\}$

<table>
<thead>
<tr>
<th>ID</th>
<th>Sequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq1</td>
<td>${a, b}, {c}, {f}, {g}, {e}$</td>
</tr>
<tr>
<td>seq2</td>
<td>${a, d}, {c}, {b}, {a, \textcolor{red}{b}}, e, f$</td>
</tr>
<tr>
<td>seq3</td>
<td>${a}, {\textcolor{red}{b}}, {f}, {e}$</td>
</tr>
<tr>
<td>seq4</td>
<td>${b}, {f, g}$</td>
</tr>
</tbody>
</table>

In this example, no item can expand the left itemset of the rule!
How to choose items for performing right expansions of a rule $X \Rightarrow Y$?

Scan the sequences containing the rule and note items appearing in at least $\text{minsup} \times |S|$ sequences after the first occurrence of $X$.

For example: $\{a\} \Rightarrow \{b\}$

<table>
<thead>
<tr>
<th>ID</th>
<th>Sequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq1</td>
<td>${a, b}, {c}, {f}, {g}, {e}$</td>
</tr>
<tr>
<td>seq2</td>
<td>${a, d}, {c}, {b}, {a, b, e, f}$</td>
</tr>
<tr>
<td>seq3</td>
<td>${a}, {b}, {f}, {e}$</td>
</tr>
<tr>
<td>seq4</td>
<td>${b}, {f, g}$</td>
</tr>
</tbody>
</table>

The following items meet these criteria:
- $c$ : seq1, seq2
- $e$ : seq1, seq2, seq3
- $f$ : seq1, seq2, seq3
How to avoid generating the same rules twice?

Problem 1: The same rule can be generated by adding items in different orders.

\[
\{a\} \Rightarrow \{b\}.
\]

\[
\{a, c\} \Rightarrow \{b\}.
\]

\[
\{a, c, d\} \Rightarrow \{b\},
\]

\[
\{a, d\} \Rightarrow \{b\}.
\]

Solution: Add only an item to the left/right part of a rule if the item is larger than all items already in the left/right part.
How to avoid generating the same rules twice?

Problem 2: the same rule can be generated by adding items in different orders of left/right expansions.

Solution: Do not allow performing a left expansion after a right expansion. But allow performing a right expansion after a left expansion.
Implementation

**Optimization 1:** The set of sequences containing $X$, $Y$, and $X \Rightarrow Y$ is kept for each rule $X \Rightarrow Y$ generated so that the confidence can be calculated efficiently.

**Optimization 2:** During the first database scan, record the first and last occurrence of each item for each sequence.

- This allows to create initial rules very efficiently.
- This allows to avoid scanning sequences completely when searching for items for expansions.
Performance Evaluation

- RuleGrowth, CMRules and CMDEO.
- Java, 1GB of RAM
- Three real-life public datasets.

<table>
<thead>
<tr>
<th></th>
<th>Kosarak</th>
<th>BMS</th>
<th>Toxin-Snake</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence count</td>
<td>70,000</td>
<td>59,601</td>
<td>163</td>
</tr>
<tr>
<td>Item count</td>
<td>21,144</td>
<td>497</td>
<td>20</td>
</tr>
<tr>
<td>Average item count by sequence</td>
<td>7.97</td>
<td>2.51</td>
<td>60.61</td>
</tr>
<tr>
<td>Average different item count by sequence</td>
<td>7.97</td>
<td>2.51</td>
<td>17.84</td>
</tr>
</tbody>
</table>
Influence of $\text{minsup}$

Snake

BMS
Influence of $\minsup$

Kosarak

- Execution Time (s)
- Rule Count
- Memory (MB)
Influence of minconf

Kosarak

BMS

Snake
Conclusion

RuleGrowth,

• Is a novel algorithm for mining sequential rules common to several sequences,
• It outperforms **CMRules** and **CMDeo** in terms of execution time and memory usage.
• Source code and datasets available as part of the **SPMF data mining library** (GPL 3).
Introduction

SPMF is an open-source data mining mining library written in Java, specialized in pattern mining.

It is distributed under the GPL v3 license.

It offers implementations of 120 data mining algorithms for:

- association rule mining,
- itemset mining,
- sequential pattern mining,
- sequential rule mining,
- sequence prediction,
- periodic pattern mining,
- high-utility pattern mining,
- clustering and classification

The source code of each algorithm can be easily integrated in other Java software.

Moreover, SPMF can be used as a standalone program with a simple user interface or from the command line.

SPMF is fast and lightweight (no dependencies to other libraries).

The current version is v0.99j and was released the 16th June 2016.

http://www.philippe-fournier-viger.com/spmf/
Running an algorithm

Choose an algorithm: CM-SPAM

Choose input file: snake_192_converted.txt

Set output file: test.txt

Choose minsup (%): 0.96 (e.g. 0.5 or 50%)

Min pattern length (optional): 4 (e.g. 1 item)

Max pattern length (optional): (e.g. 10 items)

Max gap (optional): (e.g. 1,2,3)

Required items (optional): (e.g. 1,2,3)

Show sequence ids? (optional): (default: false)

Open output file:
- using SPMF viewer
- using text editor

Run algorithm

Algorithm is running...

CM-SPAM v0.87: STATISTICS

Total time ~ 135 ms

Frequent sequences count: 447

Max memory (mb): 39.53382110595703447

minsup 157

Intersection count 2141

Discovered patterns

http://www.philippe-fournier-viger.com/spmf/
Some applications

E-learning


Some applications

Manufacturing simulation


Quality control

Some applications

Web page prefetching


Anti-pattern detection in service based systems,


Embedded systems

Some applications

Alarm sequence analysis

Recommendation
Some applications

Restaurant recommendation


Customer behavior analysis

Extensions

Extensions:

- **TRuleGrowth**: mining rules with a window size constraint
- **TopSeqRules**: mining the top-k sequential rules.
- **TNS**: mining the top-k non redundant sequential rules
- ...