A constraint-based pattern mining algorithm and its optimisation for multicore systems

Sofya Titarenko, Valeriy Titarenko, Georgios Aivaliotis, Jan Palczewski

EMiT 2019
What is pattern mining?

We can mine for interesting patterns, co-occurring patterns, frequent patterns, etc... A frequent pattern is a set of events which are met often

Eg. datasets of transactions in supermarkets, road accidents, bioinformatics, environmental, health records, etc..

Why do we mine for frequent patterns?

We often use found frequent patterns for the future analysis: clustering, building predictive models, classification, etc..
Challenges in frequent pattern mining

Storage Space

Computational time

However...

We want to solve our problem in real-time.

We want to keep it running on stand-alone workstation.

Ex: making a medical decision

Ex: working with sensitive datasets
Challenges in frequent pattern mining

Taking time into account
Ex. Internet queries, medical monitoring, environmental monitoring etc..

Allowing for uncertainty in datasets
Human error, faults in sensors, sampling errors, etc..

Additional constraints: item-based, temporal, etc..
Interested only in the patterns of a particular length
Patterns formed only from the events belonging to different categories

More complexity, longer time, data storage challenge!
Challenges in frequent pattern mining

Time and uncertainty in time-points

Time-series

Adding uncertainty

![Diagram showing time-series and adding uncertainty](image-url)
Challenges in frequent pattern mining

Temporal and item-based constraints

Ex: medical records

1 4
2 3 5
1 2 3 5
2 5
1 2 3 5

Pattern is allowed to be no longer than a certain time period

Ex: weather dataset

Munich
↓
MUC, T↑

London
LDN, T↑
LDN, T↑

Berlin
BLN, T↓

t

1 2 met twice
3 5 met once

Pattern is allowed to contain only items from different groups
Our algorithm FARPAM

Defines patterns to accommodate uncertainty, temporal and item-based constraints

Optimises calculations to be fast and efficient on a standalone multicore workstation
Steps for optimisation

Making storage more efficient

1. Clean dataset
3. For binary vectors pack all the information in 32 bit.
Steps for optimisation

Improving processing time

1. Use of bitmap vectors and therefore binary logic operators (for ex. ADD)
2. Use multithreading and vectorisation wherever possible
3. Use caching memory strategy
4. Clean dataset
5. Store events in a “clever” way
FARPAM compared with FARPAMp

FARPAMp includes prior information:

• Specifically, the duration of uncertainty intervals is the same for all events of a certain type.
• The algorithm could be modified for other prior constraints
## Optimisation Results

Weather dataset, no uncertainty. Daily measurements over few years, 25 European cities

<table>
<thead>
<tr>
<th>support</th>
<th>No. pat</th>
<th>apriori</th>
<th>apriori+opt</th>
<th>SPAM</th>
<th>FARPAM</th>
<th>FARPAMp</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>8332</td>
<td>120.1</td>
<td>18.7</td>
<td>14.7</td>
<td>1.19</td>
<td>1.21</td>
</tr>
<tr>
<td>0.4</td>
<td>46848</td>
<td>5942.1</td>
<td>51.9</td>
<td>50.4</td>
<td>3.66</td>
<td>3.87</td>
</tr>
<tr>
<td>0.3</td>
<td>157536</td>
<td>7519.8</td>
<td>120.4</td>
<td>219.4</td>
<td>7.8</td>
<td>7.85</td>
</tr>
</tbody>
</table>

Times are in seconds  

~20 times faster than SPAM without uncertainty  

810 records, 910,387 events
## Optimisation Results

18,518 records, 304,719 events (Adult social care dataset, with uncertainty)

<table>
<thead>
<tr>
<th>support</th>
<th>robustSpam</th>
<th>Apriori</th>
<th>Apriori+omp</th>
<th>FARPAM</th>
<th>FARPAMp</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>715.9</td>
<td>20.9</td>
<td>2.3</td>
<td>0.74</td>
<td>0.4</td>
</tr>
<tr>
<td>0.05</td>
<td>2370.7</td>
<td>91.7</td>
<td>8.1</td>
<td>1.06</td>
<td>1.05</td>
</tr>
<tr>
<td>0.03</td>
<td>5984.1</td>
<td>256.3</td>
<td>14.1</td>
<td>1.66</td>
<td>1.58</td>
</tr>
<tr>
<td>0.02</td>
<td>13045.1</td>
<td>602.2</td>
<td>26.8</td>
<td>2.3</td>
<td>1.88</td>
</tr>
</tbody>
</table>

~6000 times faster than robustSPAM with uncertainty

Only for pattern length [3,5]
Optimisation approaches

Events

\begin{align*}
\text{events} & : \quad A \ B \ C \ B \ A \ A \ D \ B \ C \ A \ D \\
\text{times} & : \quad t1s, \ t1e \ t2s, \ t2e \ t3s, \ t3e... \\
\end{align*}

No. of unique events

\begin{align*}
\text{events} & : \quad A \ B \ C \ D \\
\text{times} & : \quad t1s, \ t1e \ t2s, \ t2e \ t3s, \ t3e... \\
\end{align*}
Optimisation approaches

<table>
<thead>
<tr>
<th>previously found patterns</th>
<th>candidate pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>bcd</td>
<td>acd</td>
</tr>
<tr>
<td>0 &amp; 1 &amp; 1 &amp; 0 = 0</td>
<td></td>
</tr>
<tr>
<td>1 &amp; 0 &amp; 1 &amp; 1 = 0</td>
<td></td>
</tr>
<tr>
<td>1 &amp; 1 &amp; 1 &amp; 1 = 1</td>
<td></td>
</tr>
<tr>
<td>1 &amp; 1 &amp; 0 &amp; 1 = 1</td>
<td></td>
</tr>
<tr>
<td>0 &amp; 1 &amp; 1 &amp; 1 = 1</td>
<td></td>
</tr>
<tr>
<td>1 &amp; 1 &amp; 1 &amp; 1 = 1</td>
<td></td>
</tr>
<tr>
<td>1 &amp; 1 &amp; 1 &amp; 0 = 0</td>
<td></td>
</tr>
<tr>
<td>1 &amp; 1 &amp; 1 &amp; 1 = 1</td>
<td></td>
</tr>
<tr>
<td>1 &amp; 0 &amp; 1 &amp; 1 = 0</td>
<td></td>
</tr>
<tr>
<td>1 &amp; 1 &amp; 1 &amp; 0 = 1</td>
<td></td>
</tr>
</tbody>
</table>

binary vectors
Optimisation approaches

- Random selection of existing patterns
- Thread #1
  - Ordered list
    - Forming candidates based on binary vectors
      - 12 4-patterns
        - aaba
        - aabd
        - bddc
        - dbdd
      - Merging in one ordered list
        - aaba aabd aada abaa abad
        - bcda bddc dbac dbdd
    - 12 4-patterns
      - aada dbac

- Thread #2
  - Existing 3-patterns
    - Thread #3
      - 8 4-patterns
        - abaa abad bcda
Conclusions

1. When working with Big Data it is very important to use storage space carefully;
2. Algorithmic and hardware optimisation allows reduction in storage space and considerably reduces calculative time;
3. Good algorithmic formulation allows flexibility in applications and makes possible future algorithmic modifications and extensions easier.
4. Use of prior knowledge can significantly speed up calculations
Challenges

1. Datasets
Problems:
1. Weather dataset (810 records, 910,387 events)
2. LCC (18,518 records, 304,719 events)

2. Constraints, uncertainty or additional information we want to keep....

Time problem

Storage problem

Example:
1. suppose we found 2,000,000 of patterns with max length ~50. We need ~0.4 GB of memory
2. We want to accelerate problem using ID list approach. Suppose for problem 1. we have ~20,000 records. Then we need extra ~320GB

If no uncertainty (weather example)

<table>
<thead>
<tr>
<th>sup</th>
<th>N</th>
<th>pat</th>
<th>apriori</th>
<th>apriori+opt</th>
<th>SPA</th>
<th>Alg1</th>
<th>Alg2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>8332</td>
<td>120.1</td>
<td>18.7</td>
<td>14.7</td>
<td>1.19</td>
<td>1.21</td>
<td></td>
</tr>
<tr>
<td>0.4</td>
<td>46848</td>
<td>5942.1</td>
<td>51.9</td>
<td>50.4</td>
<td>3.66</td>
<td>3.87</td>
<td></td>
</tr>
<tr>
<td>0.3</td>
<td>157536</td>
<td>7519.8</td>
<td><strong>120.4</strong></td>
<td><strong>219.4</strong></td>
<td><strong>7.8</strong></td>
<td><strong>7.85</strong></td>
<td></td>
</tr>
</tbody>
</table>

Assuming that there’s no coinciding events

+uncertainty (LCC example)

<table>
<thead>
<tr>
<th>sup</th>
<th>robust Spam</th>
<th>Apriori</th>
<th>Apriori+omp</th>
<th>Alg1</th>
<th>Alg2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>715.9</td>
<td>20.9</td>
<td>2.3</td>
<td>0.74</td>
<td>0.4</td>
</tr>
<tr>
<td>0.05</td>
<td>2370.7</td>
<td>91.7</td>
<td>8.1</td>
<td>1.06</td>
<td>1.05</td>
</tr>
<tr>
<td>0.03</td>
<td>5984.1</td>
<td>256.3</td>
<td>14.1</td>
<td>1.66</td>
<td>1.58</td>
</tr>
<tr>
<td>0.02</td>
<td>13045.1</td>
<td>602.2</td>
<td>26.8</td>
<td>2.3</td>
<td>1.88</td>
</tr>
</tbody>
</table>

Only for length [3,5]

Sofya Titarenko, Valeriy Titarenko, George Aivaliotis, Jan Palczewski, “Fast implementation of pattern mining algorithms with time stamp uncertainties and temporal constraints”, to submit in journal of Big Data
The problem we solve

1. Reformulated pattern definition so to accommodate few types of pattern mining (itemset mining, time series mining, SPAM with sequences=1, time series with time uncertainty)

2. Cleaned dataset from the events which are not frequent

3. Store database in the following way: Dataset with only unique events for record, number of unique events, Dataset of times

4. Use of ID lists for frequent patterns. All ID lists are compressed in bitmap. When working with them we apply binary logic operators when it is possible

5. Check if pattern is frequent only if:
   • All its subpatterns of length (n-1) are frequent
   • Sup of the resultant logical multiplication of ID vectors is above min support value
   • Check the corresponding entry only its binary value equals 1

6. Use the following property:
   Suppose uncertainty interval is the same for alike events. If the interval starts earlier for the first event, then it also finishes earlier

7. Multithreading, vectorisation
The problem we solve

3. Store database in the following way: Dataset with only unique events for record, number of unique events, Dataset of times

<table>
<thead>
<tr>
<th>record</th>
<th>events</th>
<th>times</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A B C B</td>
<td>t1s, t1e t2s,</td>
</tr>
<tr>
<td></td>
<td>A A D B</td>
<td>t2e t3s, t3e</td>
</tr>
<tr>
<td></td>
<td>B C A D</td>
<td></td>
</tr>
</tbody>
</table>

Classical way

<table>
<thead>
<tr>
<th>record</th>
<th>events</th>
<th>times</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A B C B</td>
<td>t1s, t1e t2s,</td>
</tr>
<tr>
<td></td>
<td>A A D B</td>
<td>t2e t3s, t3e</td>
</tr>
<tr>
<td></td>
<td>B C A D</td>
<td></td>
</tr>
</tbody>
</table>

Proposed way

<table>
<thead>
<tr>
<th>record</th>
<th>events</th>
<th>times</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A B C D</td>
<td>t1s, t1e t2s,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t2e t3s, t3e</td>
</tr>
</tbody>
</table>

| N of unique events | 4 3 2 2 |

Advantages:
1. Dataset become more compact
2. Searching for pattern function works faster
The problem we solve

4. Use of ID lists for frequent patterns. All ID lists are compressed in bitmap. When working with them we apply binary logic operators when it is possible

Classical way

16 vectors for 16 records!

Proposed way

1 vector for 16 records!
If no uncertainty (weather example)

<table>
<thead>
<tr>
<th>sup</th>
<th>N pat</th>
<th>apriori</th>
<th>apriori +opt</th>
<th>SPAM</th>
<th>Alg1</th>
<th>Alg2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>8332</td>
<td>120.1</td>
<td>18.7</td>
<td>14.7</td>
<td>1.19</td>
<td>1.21</td>
</tr>
<tr>
<td>0.4</td>
<td>46848</td>
<td>5942.1</td>
<td>51.9</td>
<td>50.4</td>
<td>3.66</td>
<td>3.87</td>
</tr>
<tr>
<td>0.3</td>
<td>157536</td>
<td>7519.8</td>
<td><strong>120.4</strong></td>
<td><strong>219.4</strong></td>
<td>7.8</td>
<td>7.85</td>
</tr>
</tbody>
</table>

+uncertainty (LCC example)

<table>
<thead>
<tr>
<th>sup</th>
<th>robust Spam</th>
<th>Apriori</th>
<th>Apriori +omp</th>
<th>Alg1</th>
<th>Alg2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>715.9</td>
<td>20.9</td>
<td>2.3</td>
<td>0.74</td>
<td>0.4</td>
</tr>
<tr>
<td>0.05</td>
<td>2370.7</td>
<td>91.7</td>
<td>8.1</td>
<td>1.06</td>
<td>1.05</td>
</tr>
<tr>
<td>0.03</td>
<td>5984.1</td>
<td>256.3</td>
<td>14.1</td>
<td>1.66</td>
<td>1.58</td>
</tr>
<tr>
<td>0.02</td>
<td>13045.1</td>
<td>602.2</td>
<td>26.8</td>
<td>2.3</td>
<td>1.88</td>
</tr>
</tbody>
</table>

Assuming that there’s no coinciding events

The proposed algorithms (Alg1 and Alg2) work up to ~30 times faster then open source code SPAM and ~7,000 times faster then previously developed robustSPAM