H-Mine: Hyper-Structure Mining of Frequent Patterns in Large Databases

J. Pei, J. Han, H. Lu, S. Nishio, S. Tang, and D. Yang
Int. Conf. on Data Mining (ICDM’01), San Jose, CA

Presented by Leonid Mocofan

Paper’s goals

- Introduce a new data structure: H-struct
- Introduce a new mining algorithm: H-mine
- Introduce a new data mining methodology: space-preserving mining

Why a new algorithm?

- Two current algorithm categories:
  - Candidate generation-and-test approach:
    - E.g., Apriori algorithm
  - Pattern growth methods:
    - E.g., FP-growth, TreeProjection
- They have performance bottlenecks:
  - Huge space required for mining
  - Real databases contain all the cases
  - Large applications need more scalability

H-mine characteristics

- It has limited and precisely predictable space overhead.
- It can scale up to very large databases by using database partitioning
- When the data sets are dense, it can switch to use FP-trees to continue the mining process
Frequent pattern mining introduction

- set of items: \( I = \{x_1, \ldots, x_n\} \)
- itemset \( X \): subset of items (\( X \subseteq I \))
- transaction: \( T=(\text{tid}, X) \)
- transaction database: TBD
- support(\( X \)): number of transactions in TDB containing \( X \)

Frequent pattern mining definitions

**Frequent pattern**: For a transaction database TDB and a support threshold \( \text{min\_sup} \), \( X \) is a frequent pattern if and only if \( \text{sup}(X) \geq \text{min\_sup} \)

**Frequent pattern mining**: Finding the complete set of frequent patterns in a given transaction database with respect to a given support threshold.

H-mine algorithm

1. H-mine(Mem) – memory based, efficient pattern-growth algorithm
2. H-mine based on H-mine(Mem) for large databases by first partitioning the database
3. For dense data sets, H-mine is integrated with FP-growth dynamically

H-mine(Mem) – Example

Minimum support threshold is 2

<table>
<thead>
<tr>
<th>Trans ID</th>
<th>Items</th>
<th>Frequent-item projection</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>c,d,e,f,g,i</td>
<td>c,d,e,g</td>
</tr>
<tr>
<td>200</td>
<td>a,c,d,e,m</td>
<td>a,c,d,e</td>
</tr>
<tr>
<td>300</td>
<td>a,b,d,e,g,k</td>
<td>a,d,e,g</td>
</tr>
<tr>
<td>400</td>
<td>a,c,d,h</td>
<td>a,c,d</td>
</tr>
</tbody>
</table>

**F-list**: a-c-d-e-g

Header Table H

Frequent projections

- \( 100 \): \( a,c,d,e,g \)
- \( 200 \): \( a,c,d,E \)
- \( 300 \): \( a,d,e,g \)
- \( 400 \): \( a,c,d \)
H-mine(Mem) – Example

Header Table $H_a$ and $ac$-queue

frequent projections

100 c d e g

200 a c d g

300 a d e g

400 a c d

H-mine(Mem) – Example

Header Table $H_a$

Header Table $H_{ac}$

frequent projections

100 c d e g

200 a c d g

300 a d e g

400 a c d

H-mine(Mem) – Example

Header Table $H_a$ and $ad$-queue

frequent projections

100 c d e g

200 a c d e

300 a d e g

400 a c d

H-mine(Mem) – Example

Adjusted hyper-links after mining $a$-projected database
H-mine: Mining large databases

- TDB transaction database (size \( n \))
- Minimum support threshold \( \min\_sup \)
- Find \( L \), the set of frequent items
- TDB partitioned in \( k \) parts (\( TDB_i, 1 \leq i \leq k \))

\[
\lceil \min\_sup \ast \frac{n_i}{n} \rceil
\]

Apply H-mine(Mem) to \( TDB_i \) with minimum support threshold

Combine \( F_i \), set of locally frequent pattern in \( TDB_i \), to get the globally frequent patterns.

H-mine – Example

- TDB split in \( P_1, P_2, P_3, P_4 \)
- Minimum support threshold 100

<table>
<thead>
<tr>
<th>Local freq. pat.</th>
<th>Partitions</th>
<th>Accumulated sup. cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>ab</td>
<td>( P_1, P_2, P_3, P_4 )</td>
<td>280</td>
</tr>
<tr>
<td>ac</td>
<td>( P_1, P_2, P_3, P_4 )</td>
<td>320</td>
</tr>
<tr>
<td>ad</td>
<td>( P_1, P_2, P_3, P_4 )</td>
<td>260</td>
</tr>
<tr>
<td>abc</td>
<td>( P_1, P_2, P_3, P_4 )</td>
<td>120</td>
</tr>
<tr>
<td>abcd</td>
<td>( P_1, P_4 )</td>
<td>40</td>
</tr>
</tbody>
</table>

- Frequent patterns: ab, ac, ad, abc

Performance

- H-mine has better runtime performance on both sparse and dense data than FP-growth and Apriori
- H-mine has better space usage on both sparse and dense data than FP-growth and Apriori
- H-mine performs well with very large databases too
Conclusions

H-mine:
■ has high performance
■ is scalable in all kinds of data
■ has very small space overhead
■ can dynamically adapt to input data
■ introduces structure- and space-preserving mining methodology

Bibliography

■ “H-Mine: Hyper-Structure Mining of Frequent Patterns in Large Databases”, J. Pei, J. Han, H. Lu, S. Nishio, S. Tang, and D. Yang, Int. Conf. on Data Mining (ICDM’01), San Jose, CA, Nov. 2001.
■ “Mining Frequent Patterns without Candidate Generation”, J. Han, J. Pei, and Y. Yin, ACM-SIGMOD 2000, Dallas, TX, May 2000.