Speeding up Association rules

Dynamic Hashing and Pruning technique

Borrowed from software.ssu.ac.kr/AI08_page/class8-Association%20Rules.ppt
DHP: Reduce the Number of Candidates

- A $k$-itemset whose corresponding hashing bucket count is below the threshold cannot be frequent
  - Candidates: $a, b, c, d, e$
  - Hash entries: $\{ab, ad, ae\} \{bd, be, de\} \ldots$
  - Frequent 1-itemset: $a, b, d, e$
  - $ab$ is not a candidate 2-itemset if the sum of count of $\{ab, ad, ae\}$ is below support threshold

- J. Park, M. Chen, and P. Yu. *An effective hash-based algorithm for mining association rules*. In *SIGMOD’95*
Still challenging, the niche for DHP

- DHP (Park ’95): Dynamic Hashing and Pruning

- Candidate large 2-itemsets are huge.
  - DHP: trim them using hashing

- Transaction database is huge that one scan per iteration is costly
  - DHP: prune both number of transactions and number of items in each transaction after each iteration
How does it look like?

Apriori
- Generate candidate set
- Count support

DHP
- Generate candidate set
- Count support
- Make new hash table
Reducing Number of Comparisons

Candidate counting:
- Scan the database of transactions to determine the support of each candidate itemset.
- To reduce the number of comparisons, store the candidates in a hash structure.
  - Instead of matching each transaction against every candidate, match it against candidates contained in the hashed buckets.

<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Bread, Milk</td>
</tr>
<tr>
<td>2</td>
<td>Bread, Diaper, Beer, Eggs</td>
</tr>
<tr>
<td>3</td>
<td>Milk, Diaper, Beer, Coke</td>
</tr>
<tr>
<td>4</td>
<td>Bread, Milk, Diaper, Beer</td>
</tr>
<tr>
<td>5</td>
<td>Bread, Milk, Diaper, Coke</td>
</tr>
</tbody>
</table>
Generate Hash Tree

Suppose you have 15 candidate itemsets of length 3:

\{1 \ 4 \ 5\}, \{1 \ 2 \ 4\}, \{4 \ 5 \ 7\}, \{1 \ 2 \ 5\}, \{4 \ 5 \ 8\}, \{1 \ 5 \ 9\}, \{1 \ 3 \ 6\}, \{2 \ 3 \ 4\}, \{5 \ 6 \ 7\}, \{3 \ 4 \ 5\}, \{3 \ 5 \ 6\}, \{3 \ 5 \ 7\}, \{6 \ 8 \ 9\}, \{3 \ 6 \ 7\}, \{3 \ 6 \ 8\}

You need:

• Hash function

• Max leaf size: max number of itemsets stored in a leaf node (if number of candidate itemsets exceeds max leaf size, split the node)
Association Rule Discovery: Hash tree

Hash Function

1, 4, 7
2, 5, 8
3, 6, 9

Hash on 1, 4 or 7

Candidate Hash Tree

1 4 5
1 2 4
1 2 5
4 5 7
4 5 8

1 3 6

2 3 4
5 6 7

3 4 5
3 5 6
3 5 7
3 6 7
3 6 8

Frequent-pattern mining methods
Association Rule Discovery: Hash tree

Hash Function

Candidate Hash Tree

Frequent-pattern mining methods
Association Rule Discovery: Hash tree

Hash Function

Candidate Hash Tree

Frequent-pattern mining methods
Association Rule Discovery: Hash tree

Hash Function

1, 4, 7
2, 5, 8
3, 6, 9

Candidate Hash Tree

Hash on 1, 4 or 7
Association Rule Discovery: Hash tree

Hash on 2, 5 or 8

Hash Function

Candidate Hash Tree

Frequent-pattern mining methods
Association Rule Discovery: Hash tree

Hash Function

1,4,7
2,5,8
3,6,9

Candidate Hash Tree

Hash on 3, 6 or 9

1,4,7
2,5,8
3,6,9

1,2,4
3,5,6
4,5,7
5,6,8

1,4,5
2,3,4
3,4,5
4,5,6

1,3,6
2,5,8
3,5,7
4,5,8
5,6,7
6,8,9

1,5,9
2,3,6
3,5,8
4,5,7
6,8,9

3,6,7
3,6,8

Frequent-pattern mining methods
How to trim candidate itemsets

- In k-iteration, hash all “appearing” k+1 itemsets in a hashtable, count all the occurrences of an itemset in the correspondent bucket.

- In k+1 iteration, examine each of the candidate itemset to see if its correspondent bucket value is above the support (necessary condition)
Hash Table Construction

Consider two items sets, all items are numbered as $i_1, i_2, \ldots, i_n$. For any any pair $(x, y)$, has according to

- Hash function bucket number $\# = h(\{x, y\}) = ((\text{order of } x) \times 10 + (\text{order of } y)) \mod 7$

Example:
- Items = A, B, C, D, E,
- Order = 1, 2, 3, 4, 5,

- $H(\{C, E\}) = (3 \times 10 + 5) \mod 7 = 0$
- Thus, $\{C, E\}$ belong to bucket 0.
<table>
<thead>
<tr>
<th>TID</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>A C D</td>
</tr>
<tr>
<td>200</td>
<td>B C E</td>
</tr>
<tr>
<td>300</td>
<td>A B C E</td>
</tr>
<tr>
<td>400</td>
<td>B E</td>
</tr>
</tbody>
</table>

Figure 1. An example transaction database
### Generation of C1 & L1 (1st iteration)

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Sup</th>
</tr>
</thead>
<tbody>
<tr>
<td>{A}</td>
<td>2</td>
</tr>
<tr>
<td>{B}</td>
<td>3</td>
</tr>
<tr>
<td>{C}</td>
<td>3</td>
</tr>
<tr>
<td>{D}</td>
<td>1</td>
</tr>
<tr>
<td>{E}</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Itemset</th>
<th>Sup</th>
</tr>
</thead>
<tbody>
<tr>
<td>{A}</td>
<td>2</td>
</tr>
<tr>
<td>{B}</td>
<td>3</td>
</tr>
<tr>
<td>{C}</td>
<td>3</td>
</tr>
<tr>
<td>{E}</td>
<td>3</td>
</tr>
</tbody>
</table>

Frequent-pattern mining methods
Hash Table Construction

Find all 2-itemset of each transaction

<table>
<thead>
<tr>
<th>TID</th>
<th>2-itemset</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>{A C} {A D} {C D}</td>
</tr>
<tr>
<td>200</td>
<td>{B C} {B E} {C E}</td>
</tr>
<tr>
<td>300</td>
<td>{A B} {A C} {A E} {B C} {B E} {C E}</td>
</tr>
<tr>
<td>400</td>
<td>{B E}</td>
</tr>
</tbody>
</table>
Hash Table Construction (2)

- Hash function
  \[ h(\{x \ y\}) = ((\text{order of } x) \times 10 + (\text{order of } y)) \mod 7 \]

- Hash table

<table>
<thead>
<tr>
<th>bucket</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

- {C E}  {A E}  {B C}  {B E}  {A B}  {A C}
- {C E}  {B C}  {B E}  {C D}  {B E}  {A C}
- {A D}  

Frequent-pattern mining methods
## C2 Generation (2nd iteration)

<table>
<thead>
<tr>
<th>L1*L1</th>
<th># in the bucket</th>
</tr>
</thead>
<tbody>
<tr>
<td>{A B}</td>
<td>1</td>
</tr>
<tr>
<td>{A C}</td>
<td>3</td>
</tr>
<tr>
<td>{A E}</td>
<td>1</td>
</tr>
<tr>
<td>{B C}</td>
<td>2</td>
</tr>
<tr>
<td>{B E}</td>
<td>3</td>
</tr>
<tr>
<td>{C E}</td>
<td>3</td>
</tr>
</tbody>
</table>

Resulted C2
- {A C}
- {B C}
- {B E}
- {C E}

C2 of Apriori
- {A B}
- {A C}
- {A E}
- {B C}
- {B E}
- {C E}
Effective Database Pruning

- **Apriori**
  - Don’t prune database.
  - Prune $C_k$ by support counting on the original database.

- **DHP**
  - More efficient support counting can be achieved on pruned database.
# Performance Comparison

<table>
<thead>
<tr>
<th></th>
<th>Apriori number</th>
<th>DHP number</th>
<th>$D_{k_1}$</th>
<th>$D_{k_2}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L_1$</td>
<td>820</td>
<td>820</td>
<td>6,700KB, 100,000</td>
<td></td>
</tr>
<tr>
<td>$C_2$</td>
<td>335,790</td>
<td>338</td>
<td>6,700KB, 100,000</td>
<td></td>
</tr>
<tr>
<td>$L_2$</td>
<td>207</td>
<td>207</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$C_3$</td>
<td>618</td>
<td>618</td>
<td>659KB, 20,602</td>
<td></td>
</tr>
<tr>
<td>$L_3$</td>
<td>201</td>
<td>201</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$C_4$</td>
<td>184</td>
<td>184</td>
<td>546KB, 17,417</td>
<td></td>
</tr>
<tr>
<td>$L_4$</td>
<td>98</td>
<td>98</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$C_5$</td>
<td>30</td>
<td>30</td>
<td>332KB, 10,149</td>
<td></td>
</tr>
<tr>
<td>$L_5$</td>
<td>23</td>
<td>23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$C_6$</td>
<td>1</td>
<td>1</td>
<td>24KB, 756</td>
<td></td>
</tr>
<tr>
<td>$L_6$</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>total time</td>
<td>39.39</td>
<td>13.91</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Performance Comparison (2)

Figure 8: Execution time of Apriori and DHP
Conclusion

- Effective hash-based algorithm for the candidate itemset generation
- Two phase transaction database pruning
- Much more efficient (time & space) than Apriori algorithm