# Database Mining: Bringing Algorithms to Data

**Sharma Chakravarthy**
**I**nformation **T**echnology **Lab**oratory (IT Lab)
Computer Science and Engineering Department
The University of Texas at Arlington, Arlington, TX

Email: sharma@cse.uta.edu
URL: http://itlab.uta.edu/sharma

## Acknowledgments

- This presentation is based on the work of many of my students, especially Shiby Thomas, Mahesh Dudkiar, Hongen Zhang, Pratyush Mishra, and Himavalli Kona (and others)
- National Science Foundation and other agencies for their support



## Outline

- Data Mining and DW
- Database Mining
  - Overview
  - Spectrum
- Database Mining Architectures
  - Performance comparison
- Association Mining Using SQL-92 and SQL-OR
  - **Approaches**
  - **optimization**
  - **Performance comparison**
- Summary and Challenges



## Why Database Mining?

- Proliferation of relational database and DW necessitates mining without siphoning the data out
- Make mining to 'co-exist' with OLAP and other decision-support applications
- DM need to be a sub-process in next generation Business Intelligence (BI) Systems
- Leverage the RDBMS technology for mining
  - More than 40 years of research into RDBMSs
- Provide an integrated decision-support environment for analysts

## Data Mining Vs. Database Mining

- Data mining refers to main memory algorithms for mining
  - + **Can use arbitrary data structures**
  - + **Can optimize algorithms with proper representation (hash tree for example)**
  - - **Limited memory, need for buffer management (need to be implemented separately for each approach !)**
  - - **Data has to be siphoned out of its location (mostly from a DBMS or a Data Warehouse)**
  - - **Works well only for small data sizes (no scalability)**
  - - **Every time data is added to the DB, the process has to be repeated**
- **Solution?** *Database Mining – **Bringing algorithms to data instead of taking data to algorithms***



## Database (SQL-based) Mining: Implications

+ Leverage 4+ decades of DBMS R&D
+ Buffer management comes for free !
+ Portability due to standardization (SQL)
+ Fast development of mining algorithms
+ SMP parallelism for free with parallel database engines
+ Data not replicated outside of DBMS
+ SQL may be extended to include *ad hoc* mining queries
- However, No specialized data structures and memory management (UDF's are exception)



## Data Mining Evolution

- **File-Based** or **Main Memory** mining algorithms
  - Data mining
- **SQL-Based** mining algorithms
  - Database mining
- Parallel mining Algorithms
  - Both main memory and SQL-based
- Other approaches
  - Map/Reduce, …



## Mining Spectrum

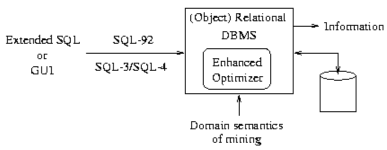
- Study architectural alternatives
- Performance evaluation
- Extend the capability of current query processors

| | | | |
|---|---|---|---|
| Cache-Mine<br>Loose Coupling | User-defined function<br>Stored Procedure | Mining extenders/blades<br>SQL-based approach | Integrated with SQL query engine |
| Mining as application on Client/app. server | Mining as application on database server | Mining using SQL+ Extensions | Integrated approach |

Loose ⟵ Integration with database ⟶ Tight

# Long-Term Vision

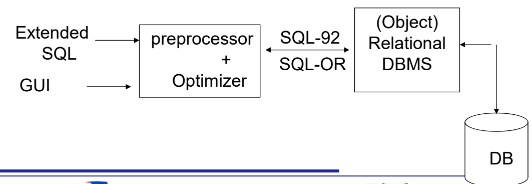
- Unbundle bulky mining operations (e.g., mine)
- Identification of Common operators
- Integration of the above into the Query Optimizer
- No distinction between OLAP and mining






# SQL-Based

- SQL: mining algorithm formulated as SQL-92/SQL-OR queries
- Several alternatives (query/subquery, Kway join)
- Can exploit SQL parallelism (where available)
- No specific extensions for mining
- May facilitate identification of needed extensions





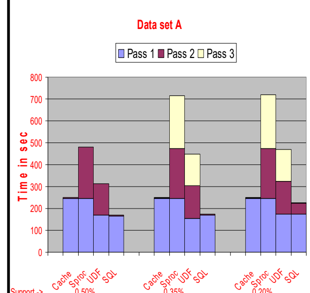

# Architecture comparison

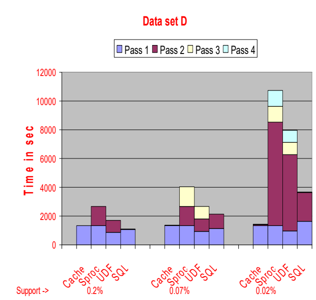
- Experiments on four real-life data sets
- Used IBM DB2 Universal Server version 5 on RS/6000: 200 MHz CPU, 256 MB main memory, 9GB disk

| Datasets | # records in millions | # Transactions in millions | # items in thousands | Avg # items |
|---|---|---|---|---|
| Dataset-A | 2.5 | 0.57 | 85 | 4.4 |
| Dataset-B | 7.5 | 2.5 | 15.8 | 2.62 |
| Dataset-C | 6.6 | 0.21 | 15.8 | 31 |
| Dataset-D | 14 | 1.44 | 480 | 9.62 |



# Architecture comparisons (DB2)



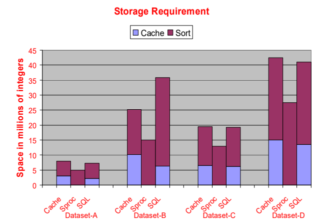
## Summary

- SQL performance good for smaller data sets
- As the size of the data set increases, SQL is not doing as good as cache (better data representation, special data structures, …)
- Stored procedure and UDF did not perform well (no optimization, limited data structures)
- SQL seems comparable with its own advantages!



## Storage Comparison

- Additional storage for Cache-Mine and SQL to cache/ transform data
- Space for indexing / sorting




- Similar storage overhead for Cache-mine and SQL





## SQL-92 based approaches to Association Rules

## Input to Mining

| TID | Item1 | Item2 | Item3 | Item4 | Item5 |
|---|---|---|---|---|---|
| 100 | 1 | | 1 | 1 | |
| 200 | | 1 | 1 | | 1 |
| 300 | 1 | 1 | 1 | | 1 |
| 400 | | 1 | | 1 | |

## Table format for database mining

| TID | ITEM |
|---|---|
| 100 | 1 |
| 100 | 3 |
| 100 | 4 |
| 200 | 2 |
| 200 | 3 |
| 200 | 5 |
| 300 | 1 |
| 300 | 2 |
| 300 | 3 |
| 300 | 5 |
| 400 | 2 |
| 400 | 5 |

- DBMSs have an upper limit on the
- Number of attributes in a table!
- Cannot use a table for horizontal layout



## Frequent itemsets table format

| $ITEM_1$ | $ITEM_2$ | $ITEM_3$ | $ITEM_4$ | $ITEM_5$ | $ITEM_6$ | $ITEM_7$ | $ITEM_8$ | NULLM | COUNT |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 2 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 3 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 3 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 3 |
| 1 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 2 |
| 2 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 2 |
| 2 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 3 |
| 3 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 2 |
| 2 | 3 | 5 | 0 | 0 | 0 | 0 | 0 | 4 | 2 |

Indicates itemset size



## Rule table format

X NULLM Y

| $ITEM_1$ | $ITEM_2$ | $ITEM_3$ | $ITEM_4$ | $ITEM_5$ | $ITEM_6$ | $ITEM_7$ | $ITEM_8$ | NULLM | RULEM | CONF | SUP |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 2 | 100 | 50 |
| 3 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 2 | 66.67 | 50 |
| 2 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 2 | 66.67 | 50 |
| 3 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 2 | 66.67 | 50 |
| 2 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 2 | 100 | 75 |
| 5 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 2 | 100 | 75 |
| 3 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 2 | 66.67 | 50 |
| 5 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 2 | 66.67 | 50 |
| 2 | 3 | 5 | 0 | 0 | 0 | 0 | 0 | 4 | 2 | 66.67 | 50 |
| 3 | 2 | 5 | 0 | 0 | 0 | 0 | 0 | 4 | 2 | 66.67 | 50 |
| 5 | 2 | 3 | 0 | 0 | 0 | 0 | 0 | 4 | 2 | 66.67 | 50 |
| 2 | 3 | 5 | 0 | 0 | 0 | 0 | 0 | 4 | 3 | 100 | 50 |
| 2 | 5 | 3 | 0 | 0 | 0 | 0 | 0 | 4 | 3 | 66.67 | 50 |
| 3 | 5 | 2 | 0 | 0 | 0 | 0 | 0 | 4 | 3 | 100 | 50 |



## SQL-based Association rule mining

- SQL-92
  - K-way joins
  - Subquery
  - 3-way joins
  - 2- group by
  - Set-oriented apriori (an improvement of K-way join)
- SQL-OR (uses table functions and other features ( blobs or binary large objects, clobs or character large objects, etc.)
  - GatherJoin
  - GatherPrune
  - GatherCount
  - Horizontal
  - vertical
  - SQL-bodied functions

## SQL-92 approaches

- Uses only the features of SQL-92 standard
  - "vanilla SQL"
  - No table functions
  - No stored procedures
  - No UDF's
- Portable (can be used on any RDBMS)

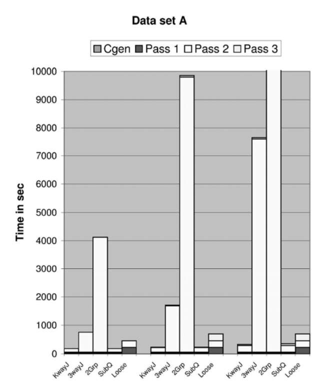


## Performance of SQL-92 approaches

- Experiments on four real-life data sets
- Used IBM DB2 Universal Server version 5 on RS/6000: 200 MHz CPU, 256 MB main memory, 9GB disk

| Datasets | # records in millions | # Transactions in millions | # items in thousands | Avg # items |
|---|---|---|---|---|
| Dataset-A | 2.5 | 0.57 | 85 | 4.4 |
| Dataset-B | 7.5 | 2.5 | 15.8 | 2.62 |
| Dataset-C | 6.6 | 0.21 | 15.8 | 31 |
| Dataset-D | 14 | 1.44 | 480 | 9.62 |






Comparison of SQL-92 approaches



## SQL-92 approaches

- Set-oriented Apriori was the overall winner
- Subquery approach performed well
- The candidate generation time and the time for first pass is much smaller than the total time
- K-way joining was also good for this data set
- As good or better than loose-coupling only for high support
- For low support considerably worse than loose-coupling
- 2-GroupBy has the worst performance
- 3-WayJoin is also not good

## Observations on apriori

- C1 (typically input relation) can be substituted by F1 for subsequent passes
- Second pass is the most expensive one; no pruning at all; it is a Cartesian product
- As the number of passes increases (i.e., longer frequent itemsets), the number of joins increases; hence materialization may be effective



## Data Characteristics

- Number of items : in thousands
- Number of Transactions: in Millions
- Data set sizes: High Gigabytes
  - Discovering all rules rather than *verifying* if a rule holds
  - Completeness and soundness
  - Performance
  - Scalability



## Input/Output Formats

- Input transaction table in the normal form
- Two attributes (tid, item)
- Example: 1: A, B, C
- Output is a collection of rules
- Rule table schema:

  ($item_1$, …, $item_k$, len, rulem, confidence, support)
- Rule AB -> CD, conf. 90%, support 5%

  (A, B, C, D, NULL, 4, 3, 0.9, 0.05)

| Tid | Item |
|---|---|
| 1 | A |
| 1 | B |
| 1 | C |



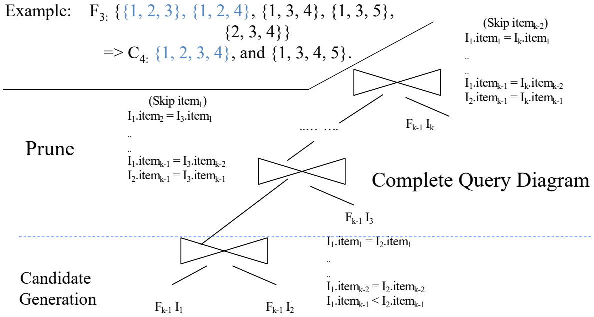
## K-way Join

- The process of support counting in Kwj is as follows:

  In any pass k:
  - Frequent itemsets of length k-1 are used to generate candidate itemsets of length k ($C_k$).
  - Prune some of the candidates generated
  - For support counting of these candidate itemsets, k copies of input relation is joined with the $C_k$.

## Candidate Set Ck Generation and pruning

Example: $F_{3:}$ {{1, 2, 3}, {1, 2, 4}, {1, 3, 4}, {1, 3, 5}, {2, 3, 4}}
=> $C_{4:}$ {1, 2, 3, 4}, and {1, 3, 4, 5}.






## Candidate Set Ck Generation

Insert into $C_k$
Select $I_1.item_1$, $I_1.item_2$, …, $I_1.item_{k-1}$, $I_2.item_{k-1}$
From $F_{k-1}$ $I_1$, $F_{k-1}$ $I_2$
Where $I_1.item_1 = I_2.item_1$ AND
$I_1.item_2 = I_2.item_2$ AND
…
…
$I_1.item_{k-2} = I_2.item_{k-2}$ AND
$I_1.item_{k-1} < I_2.item_{k-1}$

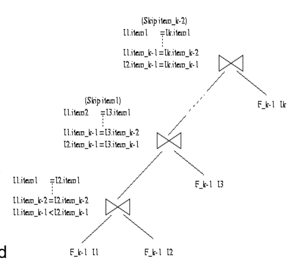
Example: $F_{3:}$ {{1, 2, 3}, {1, 2, 4}, {1, 3, 4}, {1, 3, 5}, {2, 3, 4}}
=> $C_{4:}$ {1, 2, 3, 4}, and {1, 3, 4, 5}.



## Candidate Generation in SQL

- Join step: join 2 copies of $F_{k-1}$

insert into $C_k$
select $I_1.item_1$, …, $I_1.item_{k-1}$, $I_2.item_{k-1}$
from $F_{k-1}$ $I_1$, $F_{k-1}$ $I_2$
where $I_1.item_1 = I_2.item_1$ and …. and
$I_1.item_{k-2} = I_2.item_{k-2}$ and
$I_1.item_{k-1} < I_2.item_{k-1}$




Lexicographic order of input is assumed

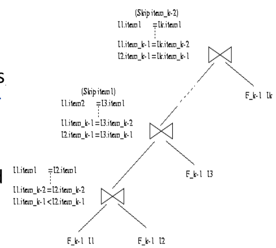


## Pruning explanation

- Consider Ck {1 3 4 5} generated from {1, 3, 4}, {1, 3, 5},
- The subsets are
  - {1 3 4} generated by skipping item at position 4
  - {1 3 5} generated by skipping item at position 3
  - {3 4 5} generated by skipping item at position 1
  - {1 4 5} generated by skipping item at position 2
- First 2 have been used in the generation of {1 3 4 5}
- Hence, skip positions 1 and 2 or 1 through k-2 (here k is 4) to check for subsets!

## Candidate Generation and Pruning

- Prune step: additional joins with (k-2) more copies of $F_{k-1}$
- Join predicates enumerated by skipping an item at a time
- k-items have k (k-1)-item subsets Out of that 2 have been used for generating the K item. No need to check them. Hence, the other (k-2) subsets need to be checked by doing (k-2) joins

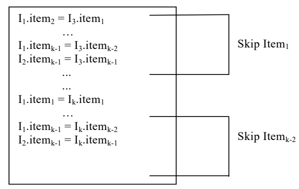



Example: $F_3$: {{1, 2, 3}, {1, 2, 4}, {1, 3, 4}, {1, 3, 5}, {2, 3, 4}}
=> $C_4$: {1, 2, 3, 4}, and {1, 3, 4, 5}.



## Pruning

Prune step: in the k-itemset of $C_k$, if there is any (k-1)-subset of $C_k$ that is not in $F_{k-1}$, we need to delete that k-itemset from $C_k$.

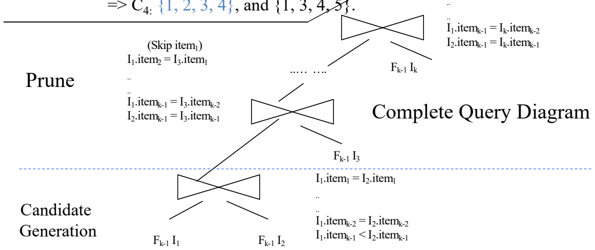



In the above example, one of the 4-itemset in $C_4$ is {1, 3, 4, 5}. This 4-itemset needs to be deleted because one of the 3-item subsets {3, 4, 5} is not in $F_3$.



## Candidate Set Ck Generation

Example: $F_3$: {{1, 2, 3}, {1, 2, 4}, {1, 3, 4}, {1, 3, 5}, {2, 3, 4}}
=> $C_4$: {1, 2, 3, 4}, and {1, 3, 4, 5}.






## SQL-92 support counting - Kway

- Join *k* copies of input table (T) with $C_k$ and do a group by on the itemsets
- insert into $F_k$

select $item_1$, … , $item_k$, count(*)
from $C_k$, T $t_1$, … , T $t_k$
where $t_1$.item = $C_k$.$item_1$ and
⋮
$t_k$.item = $C_k$.$item_k$ and
$t_1$.tid = $t_2$.tid and $t_1$.item < $t_2$.item and
⋮
$t_{k-1}$.tid = $t_k$.tid and $t_{k-1}$.item < $t_k$.item
group by $item_1$, $item_2$, … ,$item_k$
having count(*) ≥ minsup

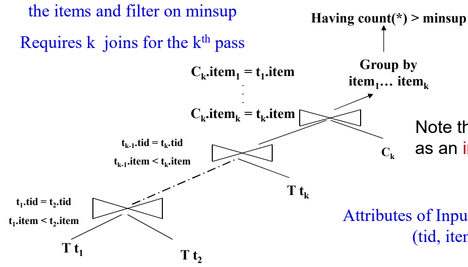
requires K more joins of $C_k$ with $T_i$ (not C or F) for the kth pass

## Support Counting for Kwj in pass k

Join $C_k$ with k copies of T

Follow up the join with a group by on the items and filter on minsup

Requires k joins for the $k^{th}$ pass



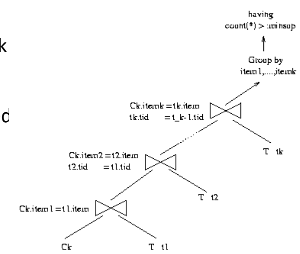

Note that Ck is used as an inner relation !

Attributes of Input Table (T) : (tid, item)



## K-way join plan ($C_k$ outer)

- Series of joins of $C_k$ with k copies of T
- Final join result is grouped on the k items






## Difference between inner and outer Ck

- Ck needs to be materialized if inner
  - Requires storage and additional I/O
- No materialization needed if outer
  - Can write a single (large) query for candidate generation, pruning, and support counting
  - May involve 10's of joins!
  - Current optimizers were not designed for that many join optimizations and importantly self-joins!



## Example: Frequent itemsets generation using Kwj

Transaction

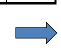
| Tid | Iid |
|---|---|
| 1 | 1 |
| 1 | 3 |
| 1 | 4 |
| 2 | 2 |
| 2 | 3 |
| 2 | 5 |
| 3 | 2 |
| 3 | 3 |
| 3 | 4 |
| 3 | 5 |
| 4 | 2 |
| 4 | 5 |

$F_1$

| Item | Count |
|---|---|
| 2 | 3 |
| 3 | 3 |
| 4 | 2 |
| 5 | 3 |

$F_1$

| Item | Coun |
|---|---|
| 2 | 3 |
| 3 | 3 |
| 4 | 2 |
| 5 | 3 |



$C_2$

| Item$_1$ | Item$_2$ |
|---|---|
| 2 | 3 |
| 2 | 4 |
| 2 | 5 |
| 3 | 4 |
| 3 | 5 |
| 4 | 5 |

join

T $t_1$

| Tid | Iid |
|---|---|
| 1 | 1 |
| 1 | 3 |
| 1 | 4 |
| 2 | 2 |
| 2 | 3 |
| 2 | 5 |
| 3 | 2 |
| 3 | 3 |
| 3 | 4 |
| 3 | 5 |
| 4 | 2 |
| 4 | 5 |

join

T $t_2$

| Tid | Iid |
|---|---|
| 1 | 1 |
| 1 | 3 |
| 1 | 4 |
| 2 | 2 |
| 2 | 3 |
| 2 | 5 |
| 3 | 2 |
| 3 | 3 |
| 3 | 4 |
| 3 | 5 |
| 4 | 2 |
| 4 | 5 |

$F_2$

| Item1 | Item2 | Count |
|---|---|---|
| 2 | 3 | 2 |
| 2 | 5 | 3 |
| 3 | 4 | 2 |
| 3 | 5 | 2 |

$C_3$

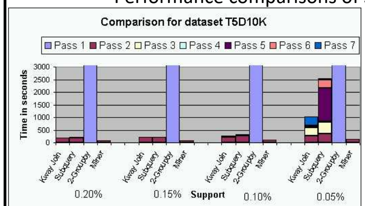
| Item1 | Item2 | Item3 |
|---|---|---|
| 2 | 3 | 5 |
| 3 | 4 | 5 |

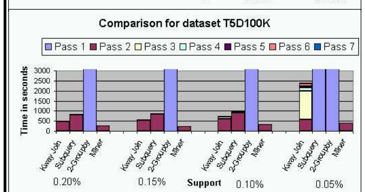
## SQL-92 support counting - Kway

- Simple and easy to write in SQL
- Only down side is the number of joins
- Also, Optimizers were never stressed for so many joins
- Optimizers were never optimized for self-joins!
- SQL is generated by a script!



## Performance comparisons of SQL-92 approaches




In both the datasets, Kway join emerged to be the winner.

For lower supports on larger datasets, pass two is the most time consuming

Intelligent Miner is not SQL based and hence appears to be faster.



## Scale-Up Experiment

Both Kway Join and Subquery approaches have the similar scale-up behavior.

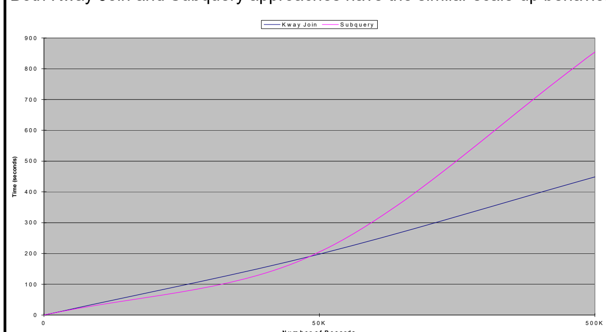



Figure 7.21 Scale-Up Experiments for SQL-92 Approaches

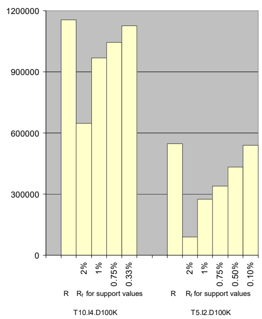


## Optimizations

- Reduce the size of input dataset
  - Non-frequent 1-itemsets are pruned out from the input table and this pruned input table is used instead in further passes.
  - Effective for higher supports (why?)
- Optimize the second pass.
  - Skip generation of $F_1$ and $C_2$ and directly generate $F_2$ by joining 2 copies of input dataset.
  - Effective for large data sets (why?)
- Reduce the number of joins done in any pass
  - Materialize all the frequent itemsets contained in any transaction at the end of the pass k and use them for support counting in pass k+1
  - Effective for higher iterations (why?)

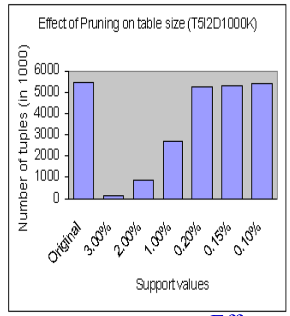
## Non-frequent item pruning

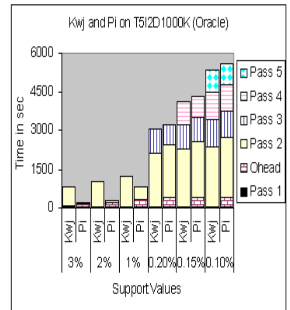
- Reducing the size of T
- T is stored in normal form
- Simply drop the (tid, item) records of non-frequent items
- Join T with $F_1$
- Significant reduction in size of T
- Improved performance especially for higher support






## Experiments

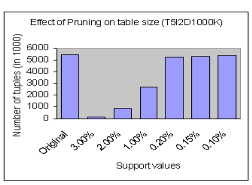


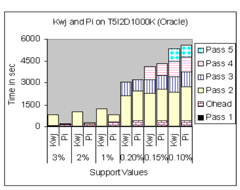



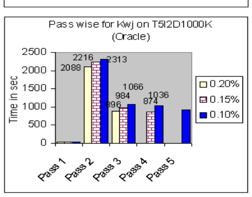

Effect of Pruning

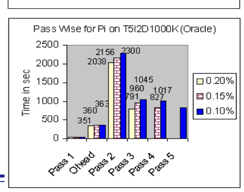


## Effect of Pruned Input



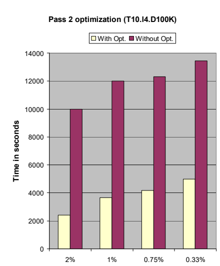






## Second pass optimization

- $C_2$ is Cartesian product of $F_1$'s
- Usually $C_2$ is very large
- Avoid generating $C_2$
- $F_2$ is found by joining 2 copies of the pruned T
- Significant performance gains

**insert** into F2
**select** p.item, q.item,count(*)
**from** Tf p, Tf q
**where** p.tid=q.tid and p.item <q.item
**group by** p.item, q.item
**having** count(*) > :minsup



## Number of Candidate Itemsets in Different Passes

| | $C_2$ | $C_3$ | $C_4$ | $C_5$ | $C_6$ | $C_7$ | $C_8$ | $C_9$ |
|---|---|---|---|---|---|---|---|---|
| T5I2D500K. Sup = 0.10% | 307720 | 126 | 7 | 0 | -- | -- | -- | -- |
| T5I2D1000K. Sup = 0.10% | 309291 | 127 | 61 | 0 | -- | -- | -- | -- |
| T10I4D100K. Sup = 0.75% | 12470 | 65 | 3 | 0 | -- | -- | -- | -- |
| T10I4D100K. Sup = 0.33% | 216153 | 2453 | 905 | 354 | 109 | 20 | 2 | 0 |



## Summary

- Explored SQL-aware implementations of association rule mining
- Analyzed the best SQL-92 option
- Cost formulae to characterize execution time
- Identified optimizations of
  - Set-oriented apriori approach
- Performance experiments and scale-up properties
- Moves us towards our short term vision
- Basis for optimizations for the long term vision



ITLAB @ CSE.UTA

## Results

CSE@UTA

| Table Name | Ranking | Supp = 0.2% | Supp = 0.15% | Supp = 0.1% | |
|---|---|---|---|---|---|
| T5I2D100K | First | RicSpo | RicSpo | Kwj | |
| | Second | All | All | RicSpo | |
| | Last | RicPi | RicPi | RicPi | |
| T5I2D500K | First | RicSpo | RicSpo | Spo | |
| | Second | Spo | Spo | RicSpo | |
| | Last | RicPi | RicPi | RicPi | |
| | Ranking | Supp = 2.0% | Supp = 1.0% | Supp = 0.75% | Supp = 0.33% |
| T10I4D100K | First | All | RicSpo | RicSpo | Ric |
| | Second | Pi | All | All | Spo |
| | Last | Ric | RicPi | RicPi | RicSpo |

Trends in Oracle for SQL-92 based approaches

All: all optimizations
RicPi: Reuse of item combinations with pruned input
RicSpo: Reuse of item combinations with Second pass optimization
Spo: Second pass optimization
Kwj: K-way join



ITLAB @ CSE.UTA

## Results

CSE@UTA

| Table Name | Ranking | Supp = 0.2% | Supp = 0.15% | Supp = 0.1% |
|---|---|---|---|---|
| T5I2D100K | First | RicSpo | Spo | RicSpo |
| | Second | Spo | RicSpo | All |
| | Last | RicPi | SpoPi | SpoPi |
| T5I2D500K | First | Spo | Spo | Spo |
| | Second | RicSpo | RicSpo | RicSpo |
| | Last | SpoPi | SpoPo | SpoPi |
| | Ranking | Supp = 2.0% | Supp = 1.0% | Supp = 0.75% |
| T10I4D100K | First | Spo | RicSpo | RicSpo |
| | Second | RicSpo | All | All |
| | Last | Ric | Kwj | Kwj |

Trends in IBM DB2/UDB for SQL-92 based approaches

All: all optimizations
RicPi: Reuse of item combinations with pruned input
RicSpo: Reuse of item combinations with Second pass optimization
Spo: Second pass optimization
Kwj: K-way join

## Mining-aware Optimizer

- Typically, data is stored in different DBMSs
- How can we perform mining on any RDBMS
- Our experiments indicated that different RDBMSs optimize queries in different ways
  - Even support variations had impact on the performance in different DBMSs
- Hence a global approach to mining did not seem appropriate !

- Analyze sql-92 and sql-or to generate and consolidate heuristics as metadata to be used by a mining-aware optimizer



## Motivation

- **Limitation:** Existing mining tools can't connect to multiple Database Management Systems.
  - **Solution**: Use Java Database Connectivity (JDBC)
- **Limitation:** Most of the mining tools use Cache-Mine architecture. Data are copied into the local disk.
  - **Solution**: Use SQL-based approach. Three of the approaches are based purely on SQL-92 and three of them are based on SQL-OR (Oracle).
- **Limitation:** Existing mining products do not provide expressive rule visualization
  - **Solution**: We use "rule-item" relationship in the association rule visualization to replace the "item-item" relationship [MINESET] or directed graph [MINER].
  - Java 3D, a new feature provided by JDK1.2, is used to implement three-dimensional display.



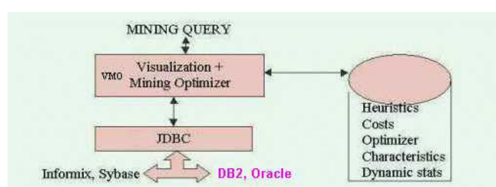
## Motivation

- **Limitation**: Most of the available products can only use the data from one table (DBMiner has 64k transactions)
  - **Solution**: We provide the user with an interface to choose the tables to be used as data source. For each table, the user can specify the columns that correspond to items. A set of JOIN/UNION operations are transparently applied to generate the input data set.
- **Limitation**: Existing products use only one mining algorithm. However, the choice of an algorithm needs to be based on data as well as DBMS characteristics.
  - **Solution**: Implement a Mining Optimizer based on meta data to decide the algorithm to be used based on the data set and the underlying DBMS used.



## Short-term Goal

- Layered architecture




- JDBC provides the database connection and SQL interface
- VMO generates and visualizes the association rules

## Mapping

- Mining is done on a relation with 2 attributes (Tid, Item)
- However, user has data in relations and has to map it into integer (Tid, Item) format
- Most mining tools accept single Tid and single column items.
- Our mining optimizer accepts multiple Tid columns and Single/Multiple Items (attributes) specified by the user from multiple relations

- Table input(Date, CustomerID, $item_1$, $item_2$, $item_3$)



## Mapping (Contd.)

InputTable1

| Date | CustID | ITEM |
|---|---|---|
| 1/1/00 | 100 | Milk |
| 1/1/00 | 100 | Eggs |
| 1/1/00 | 100 | Bread |
| 1/2/00 | 200 | Sugar |
| 1/2/00 | 200 | Eggs |
| 1/2/00 | 200 | Cake |

InputTable2

| Date | CustID | ITEM |
|---|---|---|
| 1/3/00 | 300 | Milk |
| 1/3/00 | 300 | Sugar |
| 1/3/00 | 300 | Eggs |
| 1/3/00 | 300 | Cake |
| 1/4/00 | 400 | Sugar |
| 1/4/00 | 400 | Cake |

MappedTidsTable

| Number (TIDD1) | CustID (TIDD2) | TIDI |
|---|---|---|
| 1/1/00 | 100 | 1 |
| 1/2/00 | 200 | 2 |
| 1/3/00 | 300 | 3 |
| 1/4/00 | 400 | 4 |

MappedItemsTable

| ITEMD | ITEMI |
|---|---|
| Bread | 1 |
| Cake | 2 |
| Eggs | 3 |
| Milk | 4 |
| Sugar | 5 |

FinalInputTable

| TID | ITEM |
|---|---|
| 1 | 1 |
| 1 | 3 |
| 1 | 4 |
| 2 | 2 |
| 2 | 3 |
| 2 | 5 |
| 3 | 2 |
| 3 | 3 |
| 3 | 4 |
| 3 | 5 |
| 4 | 2 |
| 4 | 5 |



## Rules table with descriptions mapped back

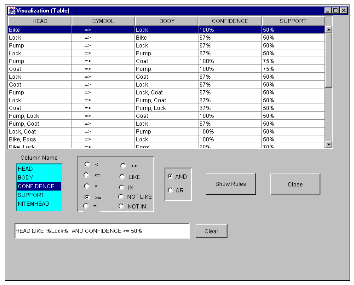
RULES (FINAL)

| Rule Head | Symbol | Rule Body | Confidence(%) | Support(%) |
|---|---|---|---|---|
| Cake | => | Eggs | 67 | 50 |
| Eggs | => | Cake | 67 | 50 |
| Eggs | => | Milk | 67 | 50 |
| Milk | => | Eggs | 100 | 50 |
| Cake | => | Sugar | 100 | 75 |
| Sugar | => | Cake | 100 | 75 |
| Eggs | => | Sugar | 67 | 50 |
| Sugar | => | Eggs | 67 | 50 |
| Cake | => | Eggs, | 67 | 50 |
| Eggs | => | Cake, | 67 | 50 |
| Sugar | => | Cake, | 67 | 50 |
| Cake, Eggs | => | Sugar | 100 | 50 |
| Cake, Sugar | => | Eggs | 67 | 50 |
| Eggs, Sugar | => | Cake | 100 | 50 |



## Rule Visualization

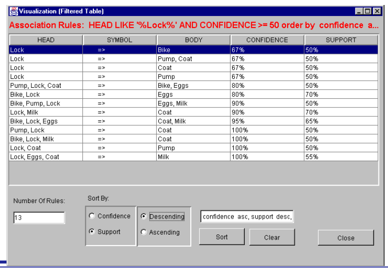
Rule Table with *Filter* capability



The key is to construct a *where* clause using the standard SQL operators, such as 'LIKE', 'NOT', 'IN', 'AND', etc

## Rule Visualization

Rule Table with *Sort* capability



Visualization (Filtered Table)

Association Rules: HEAD LIKE '%Lock%' AND CONFIDENCE >= 50 order by confidence a...

| HEAD | SYMBOL | BODY | CONFIDENCE | SUPPORT |
|---|---|---|---|---|
| Lock | => | Bike | 67% | 50% |
| Lock | => | Pump, Coat | 67% | 50% |
| Lock | => | Coat | 67% | 50% |
| Lock | => | Pump | 67% | 50% |
| Pump, Lock, Coat | => | Bike, Eggs | 80% | 50% |
| Bike, Lock | => | Eggs | 80% | 70% |
| Bike, Pump, Lock | => | Eggs, Milk | 90% | 50% |
| Lock, Milk | => | Coat | 90% | 70% |
| Bike, Lock, Eggs | => | Coat, Milk | 95% | 65% |
| Pump, Lock | => | Coat | 100% | 50% |
| Bike, Lock, Milk | => | Coat | 100% | 50% |
| Lock, Coat | => | Pump | 100% | 50% |
| Lock, Eggs, Coat | => | Milk | 100% | 55% |

Number Of Rules: 13

Sort By: Confidence / Support — Descending / Ascending

confidence asc, support desc,

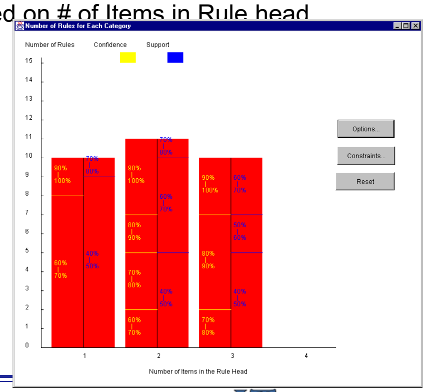
Sort | Clear | Close



## Rule Visualization

# of Rules based on # of Items in Rule head

- Customize the graph
  - by modifying the attributes
- Add more constraints
  - by modifying the *where* clause



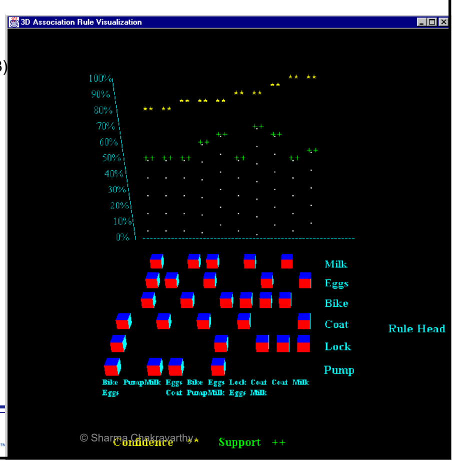



3-D Visualization
Obtained by clicking
On the previous (head:3)

All database access using relational operators

Each column is a rule

1st coat, lock, pump
-> bike, eggs
(50% sup, 80% conf)

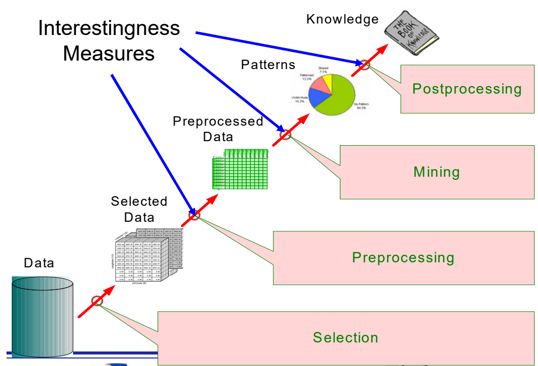





## Issues with Large Set of Rules

- Even a small data set can produce hundreds and thousands of rules (depending on the support used)
  - How to deal with them?
  - How do we prune rules systematically
- Objective interestingness measures
  - Support, confidence, and correlation
  - Used to rank patterns (itemsets or rules)
  - Analyze top-k patterns
- Subjective interestingness measures
  - {butter} →{bread} is subjectively *not* interesting
  - {Diaper} → {Beer} is subjectively interesting

## Application of Interestingness Measure




## Computing Interestingness Measure

- Given a rule X → Y, information needed to compute rule interestingness can be obtained from a contingency table

2-way Contingency table for X → Y

| | Y | $\bar{Y}$ | |
|---|---|---|---|
| X | $f_{11}$ | $f_{10}$ | $f_{1+}$ |
| $\bar{X}$ | $f_{01}$ | $f_{00}$ | $f_{o+}$ |
| | $f_{+1}$ | $f_{+0}$ | \|T\| |

$f_{11}$: support of X and Y
$f_{10}$: frequency of X and $\bar{Y}$
$f_{01}$: support of $\bar{X}$ and Y
$f_{00}$: support of $\bar{X}$ and $\bar{Y}$

f1+ : support count for X

F+1: support count for Y

Used to define various measures

- support, confidence, lift, Gini, J-measure, etc.

## Drawback of Confidence
## (people who drink coffee and tea)

| | Coffee | $\overline{\text{Coffee}}$ | |
|---|---|---|---|
| Tea | 15 | 5 | 20 |
| $\overline{\text{Tea}}$ | 75 | 5 | 80 |
| | 90 | 10 | 100 |

Independently gathered information
Use it evaluate the following rule

Association Rule: Tea → Coffee
support is 15/100 or 15% (pretty high)

Confidence= P(Coffee|Tea) = 15/20 = 0.75 = 75% (high)

but P(Coffee) = 90/100 = 0.9 (people who drink coffee regardless of whether they drink Tea)

⇒ Although confidence is high, rule is misleading

⇒ P(Coffee|$\overline{\text{Tea}}$) = 75/80 = 0.9375 (people who drink coffee and not tea)

## Statistical Independence

- Population of 1000 students
  - 600 students know how to swim (S)
  - 700 students know how to bike (B)
  - 420 students know how to swim and bike (S,B)
  - $P(S \wedge B) = 420/1000 = 0.42$
  - $P(S) \times P(B) = 0.6 \times 0.7 = 0.42$
  - $P(S \wedge B) = P(S) \times P(B)$ => Statistical independence
  - $P(S \wedge B) > P(S) \times P(B)$ => Positively correlated
  - $P(S \wedge B) < P(S) \times P(B)$ => Negatively correlated

## Statistical-based Measures

- Measures that take into account statistical (in)dependence
- PS is Pietesky-Shapiro measure

$$Lift = \frac{P(Y|X)}{P(Y)}$$

$$Interest = \frac{P(X,Y)}{P(X)P(Y)}$$

$$PS = P(X,Y) - P(X)P(Y)$$

$$\phi - coefficient = \frac{P(X,Y) - P(X)P(Y)}{\sqrt{P(X)[1-P(X)]P(Y)[1-P(Y)]}}$$

## Example: Lift/Interest

| | Coffee | $\overline{\text{Coffee}}$ | |
|---|---|---|---|
| Tea | 15 | 5 | 20 |
| $\overline{\text{Tea}}$ | 75 | 5 | 80 |
| | 90 | 10 | 100 |

Association Rule: Tea → Coffee

Confidence= P(Coffee|Tea) = 0.75

but P(Coffee) = 0.9

⇒ Lift = 0.75/0.9= 0.8333 (< 1, therefore is negatively associated)

## Drawback of Lift & Interest

| | Y | $\overline{Y}$ | |
|---|---|---|---|
| X | 10 | 0 | 10 |
| $\overline{X}$ | 0 | 90 | 90 |
| | 10 | 90 | 100 |

$$Lift = \frac{0.1}{(0.1)(0.1)} = 10$$

| | Y | $\overline{Y}$ | |
|---|---|---|---|
| X | 90 | 0 | 90 |
| $\overline{X}$ | 0 | 10 | 10 |
| | 90 | 10 | 100 |

$$Lift = \frac{0.9}{(0.9)(0.9)} = 1.11$$

**Statistical independence:**

**If P(X,Y)=P(X)P(Y) => Lift = 1**

**There are lots of measures proposed in the literature**

**Some measures are good for certain applications, but not for others**

**What criteria should we use to determine whether a measure is good or bad?**

**What about Apriori-style support based pruning? How does it affect these measures?**

| # | Measure | Formula |
|---|---|---|
| 1 | $\phi$-coefficient | $\frac{P(A,B)-P(A)P(B)}{\sqrt{P(A)P(B)(1-P(A))(1-P(B))}}$ |
| 2 | Goodman-Kruskal's ($\lambda$) | $\frac{\sum_j \max_k P(A_j,B_k)+\sum_k \max_j P(A_j,B_k)-\max_j P(A_j)-\max_k P(B_k)}{2-\max_j P(A_j)-\max_k P(B_k)}$ |
| 3 | Odds ratio ($\alpha$) | $\frac{P(A,B)P(\overline{A},\overline{B})}{P(A,\overline{B})P(\overline{A},B)}$ |
| 4 | Yule's $Q$ | $\frac{P(A,B)P(\overline{A}\overline{B})-P(A,\overline{B})P(\overline{A},B)}{P(A,B)P(\overline{A}\overline{B})+P(A,\overline{B})P(\overline{A},B)} = \frac{\alpha-1}{\alpha+1}$ |
| 5 | Yule's $Y$ | $\frac{\sqrt{P(A,B)P(\overline{A}\overline{B})}-\sqrt{P(A,\overline{B})P(\overline{A},B)}}{\sqrt{P(A,B)P(\overline{A}\overline{B})}+\sqrt{P(A,\overline{B})P(\overline{A},B)}} = \frac{\sqrt{\alpha}-1}{\sqrt{\alpha}+1}$ |
| 6 | Kappa ($\kappa$) | $\frac{P(A,B)+P(\overline{A},\overline{B})-P(A)P(B)-P(\overline{A})P(\overline{B})}{1-P(A)P(B)-P(\overline{A})P(\overline{B})}$ |
| 7 | Mutual Information ($M$) | $\frac{\sum_i \sum_j P(A_i,B_j)\log\frac{P(A_i,B_j)}{P(A_i)P(B_j)}}{\min(-\sum_i P(A_i)\log P(A_i), -\sum_j P(B_j)\log P(B_j))}$ |
| 8 | J-Measure ($J$) | $\max\Big(P(A,B)\log(\frac{P(B\|A)}{P(B)}) + P(A\overline{B})\log(\frac{P(\overline{B}\|A)}{P(\overline{B})}),$ $P(A,B)\log(\frac{P(A\|B)}{P(A)}) + P(\overline{A}B)\log(\frac{P(\overline{A}\|B)}{P(\overline{A})})\Big)$ |
| 9 | Gini index ($G$) | $\max\Big(P(A)[P(B\|A)^2 + P(\overline{B}\|A)^2] + P(\overline{A})[P(B\|\overline{A})^2 + P(\overline{B}\|\overline{A})^2]$ $-P(B)^2 - P(\overline{B})^2,$ $P(B)[P(A\|B)^2 + P(\overline{A}\|B)^2] + P(\overline{B})[P(A\|\overline{B})^2 + P(\overline{A}\|\overline{B})^2]$ $-P(A)^2 - P(\overline{A})^2\Big)$ |
| 10 | Support ($s$) | $P(A,B)$ |
| 11 | Confidence ($c$) | $\max(P(B\|A), P(A\|B))$ |
| 12 | Laplace ($L$) | $\max\Big(\frac{NP(A,B)+1}{NP(A)+2}, \frac{NP(A,B)+1}{NP(B)+2}\Big)$ |
| 13 | Conviction ($V$) | $\max\Big(\frac{P(A)P(\overline{B})}{P(A\overline{B})}, \frac{P(B)P(\overline{A})}{P(B\overline{A})}\Big)$ |
| 14 | Interest ($I$) | $\frac{P(A,B)}{P(A)P(B)}$ |
| 15 | cosine ($IS$) | $\frac{P(A,B)}{\sqrt{P(A)P(B)}}$ |
| 16 | Piatetsky-Shapiro's ($PS$) | $P(A,B) - P(A)P(B)$ |
| 17 | Certainty factor ($F$) | $\max\Big(\frac{P(B\|A)-P(B)}{1-P(B)}, \frac{P(A\|B)-P(A)}{1-P(A)}\Big)$ |
| 18 | Added Value ($AV$) | $\max(P(B\|A) - P(B), P(A\|B) - P(A))$ |
| 19 | Collective strength ($S$) | $\frac{P(A,B)+P(\overline{A}\overline{B})}{P(A)P(B)+P(\overline{A})P(\overline{B})} \times \frac{1-P(A)P(B)-P(\overline{A})P(\overline{B})}{1-P(A,B)-P(\overline{A}\overline{B})}$ |
| 20 | Jaccard ($\zeta$) | $\frac{P(A,B)}{P(A)+P(B)-P(A,B)}$ |
| 21 | Klosgen ($K$) | $\sqrt{P(A,B)}\max(P(B\|A) - P(B), P(A\|B) - P(A))$ |

## Discussion

12/13/2020 © Sharma Chakravarthy 73

## Thank You !!!

For more information visit:
http://itlab.uta.edu

Spring 2019 CSE 6331