Efficient Discovery of Concise Association Rules from Large Databases

Vikram Pudi
vikram@iiit.ac.in
IIIT Hyderabad

Talk Outline

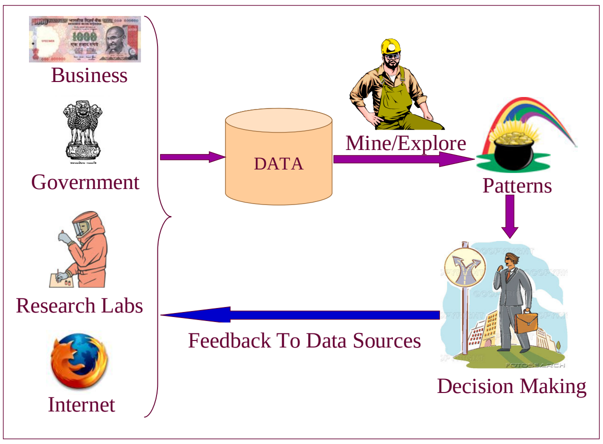
- Introduction
- Mining Association Rules
- Conciseness of Mining Results
- Conclusions

Talk Outline

- Introduction
  - Define Association Rules
  - Applications
  - Types of Association Rules
  - Interestingness Measures
  - Privacy
- Mining Association Rules
- Conciseness of Mining Results
- Conclusions

What is Data Mining?

Automated extraction of interesting patterns from large databases



Types of Patterns

- Associations
  - *Coffee* buyers usually also purchase *sugar*
- Sequence Patterns
  - After seeing *Superman*, people usually see *Star Wars*
- Clustering
  - Segments of customers requiring different promotion strategies
- Classification
  - Customers expected to be *loyal*

Association Rules

D :

Transaction ID	Items
1	Tomato, Potato, Onions
2	Tomato, Potato, Brinjal, Pumpkin
3	Tomato, Potato, Onions, Chilly
4	Lemon, Tamarind

Rule: Tomato, Potato → Onion (confidence: 66%, support: 50%)

Support(X) = |transactions containing X| / |D|

Confidence(R) = support(R) / support(LHS(R))

Problem proposed in [AIS 93]: Find all rules satisfying user given minimum support and minimum confidence.

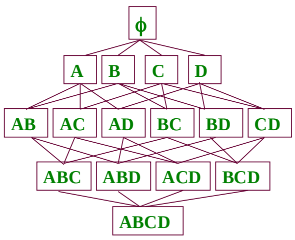
7

Typical Solution Strategy

- STEP 1: Find all *frequent* itemsets (computationally expensive)
  - Itemset X is frequent iff *support*(X) ≥ *minsup*
- STEP 2: Find *rules* from the frequent itemsets (computationally inexpensive)
  - Rule quantity: *too many rules* are usually generated
  - Rule quality: not all rules are *interesting*

8

Difficulty



- Extremely computationally expensive
- Naïve solution
  - exponential time and memory w.r.t. |I|
  - linear time w.r.t. |D|
- Typically, |I| is in thousands, |D| is in billions…

9

Applications

- E-commerce
  - People who have bought *Sundara Kandam* have also bought *Srimad Bhagavatham*
- Census analysis
  - *Immigrants* are usually *male*
- Sports
  - A chess end-game configuration with "*white pawn on A7*" and "*white knight dominating black rook*" typically results in a "*win for white*".
- Medical diagnosis
  - Allergy to *latex rubber* usually co-occurs with allergies to *banana* and *tomato*

Killer Apps

Classification

Idea: Model of each class consists of frequent itemsets for that class. Compare new transactions with each model and select "nearest" one.

Advantages: Scales well to thousands of attributes and billions of rows.

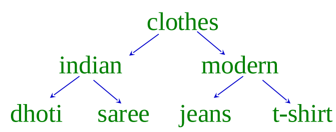
Recommendation Systems

- People who listen to songs that you listen, have also listened to these other songs…
- People who have bought these books, have also bought these other books…

11

Types of Association Rules

- Boolean association rules
- Hierarchical rules



dhoti, saree → t-shirt

- Quantitative & Categorical rules
  - (Age: 30…39), (Married: Yes) → (NumCars: 2)

12

More Types of Association Rules

- Cyclic / Periodic rules
  - Sunday → vegetables
  - Christmas → gift items
  - Summer, rich, jobless → ticket to Hawaii
- Constrained rules
  - Show itemsets whose average price > Rs.10,000
  - Show itemsets that have television on RHS
- Sequential rules
  - Star wars, Empire Strikes Back → Return of the Jedi

Interestingness Measures

Traditional Measures

- Confidence: Likelihood of a rule being true
- Support:
  - Statistical significance: Data supports rule
  - Applicability: Rule with high support is applicable in large number of transactions

Problem with Confidence

- Researcher → Coffee (confidence: 90%)
- This rule intuitively means:
  - Researchers have a *strong tendency* to drink coffee.
- But if 95% of general population drinks coffee, then this is a bad rule.
- Solution: For, X → Y,
  - Interest = P(X,Y) / P(X) P(Y)
  - Conviction = P(X) P(¬Y) / P(X, ¬Y)
    Reason: X → Y ⇔ ¬X V Y ⇔ ¬(X ^ ¬Y)

Surprising Patterns

- An itemset is uninteresting if its support can be estimated based on:
  - supports of its subsets
  - its support at earlier points in time
- Key Idea:
  - For an itemset *X*, remove items in each transaction that are not in *X*
  - If resulting database can be *compressed* well, then *X* is uninteresting (as it encodes less information)

Current Status of Interestingness

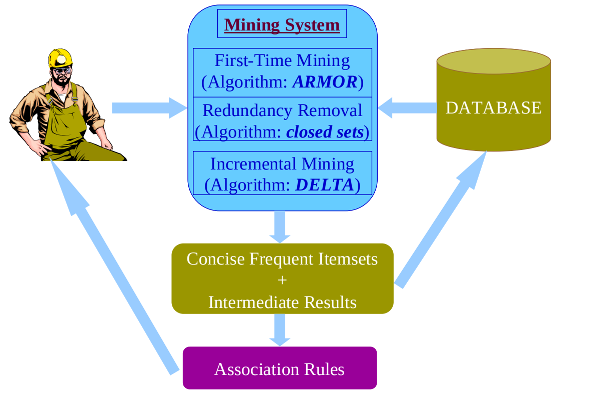
- *minsup* is required.
- So, get frequent itemsets first.
- Other interestingness measures can be applied later.
- Open Problem: How to select a good minimum support?

Issue: Privacy

- Users provide inaccurate data to protect their privacy.
- How can inaccurate data be effectively mined?
- How can data be modified in such a way as to ensure data privacy and rule accuracy?
- How can data be modified in such a way as to ensure rule privacy? – hide sensitive rules
- Can mined results be used to retrieve original data transactions?

19

Architecture for Association Rule mining

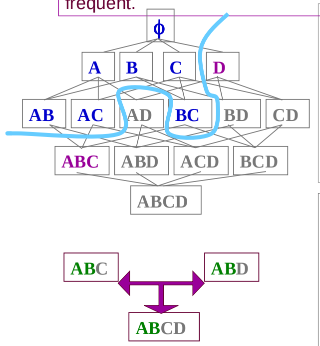


Talk Outline

- Introduction
- Mining Association Rules
  - Apriori
  - Partition
  - Sampling
  - Incremental Mining
  - FP-Growth (Frequent Pattern Growth)
  - Optimal Infeasible Algorithm: Oracle
  - ARMOR (Association Rule Mining Based on ORacle)
- Conciseness of Mining Results
- Conclusions

The Apriori Algorithm

Idea: An itemset can be frequent only if all its subsets are frequent.




```
Apriori( DB, minsup ):
C = {all 1-itemsets}
        // candidates = singletons
while ( |C| > 0 ):
    make pass over DB, find counts of C
    F = sets in C with count ≥ minsup*|DB|
    output F
    C = AprioriGen(F) // gen. candidates
```

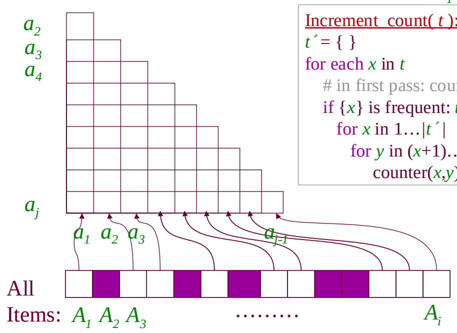
```
AprioriGen( F ):
for each pair of itemsets X, Y in F:
   if X and Y share all items, except last
      Z = X ∪ Y // generate candidate
      if any imm. subset of Z is not in F:
         prune Z // Z can't be frequent
```

22

Data Structures: For 1&2-itemsets

Transaction, t: $b_1 b_2 b_3 b_4 \ldots b_r$

```
Increment_count( t ): # second pass
t´ = { }
for each x in t
   # in first pass: counter(x)++
   if {x} is frequent: t´ = t´ ∪ {x}
      for x in 1…|t´|
         for y in (x+1)…|t´|
            counter(x,y) ++
```

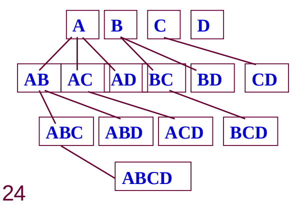


23

For Itemsets with length > 2

- Original Apriori uses *hash-trees*
  - Following *forest* structure (see figure) is as effective, and much simpler
  - Each *leaf* node has a *counter*
- Let current transaction t = ACD. Clearly, no child of B can be in t; So, the *search space is pruned*

Tree Structure




```
Increment_count( t ):
for each item x in t  # e.g. x = A,C,D
   n = root node cp. to x # n = A,C,D
   t_v = bit-vector cp. to t # e.g. 1011
   Increment_count(n, t_v)
Increment_count( n, t_v ): #e.g. n = A
# recursive depth-first search (DFS)
if n is a leaf:  n.count ++
for each child m of n  # AB,AC,AD
  if item m - n ∈ t_v  # m - n = B, C, D
    Increment_count(m, t_v)
```

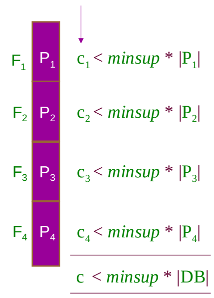
24

Analysis of Apriori

- ✓ Minimum number of candidates possible (more or less)
- ✗ i/o intensive: too many database scans
- ✗ cpu intensive: counting technique traverses data-structures for each transaction

The Partition Algorithm

Count of some itemset in each partition



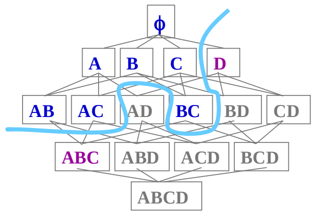
Cannot be frequent overall.

- Conceptually partition horizontally
- Pass 1: Mine each partition separately with same relative *minsup*. E.g. Use Apriori for this.
- Club individual mining results
- Pass 2: Check if clubbed results are frequent in whole database
- An itemset infrequent in each partition will be infrequent.

Analysis of Partition

- ✓ Only two i/o scans over database
- ✗ Frequent itemset mining is mostly cpu-intensive
- ✗ Partition *is* cpu intensive
  - ✗ same counting technique as Apriori
  - ✗ Number of candidates in both scans similar to that of Apriori
  - ✗ So, does double the work
- ✗ Starts fresh Apriori for each partition instead of using previous partitions' results
- ✗ Sensitive to *skew* in data distribution
  - ✗ Some partitions may be very different and can contribute spurious candidates

Sampling



Idea: Mine a random sample *S* (of *DB*) to get F_S and N_S and treat F_S as an *estimate* of *F*

Negative Border N_S (of F_S): *Minimally infrequent* itemsets

- ⇒ Infrequent candidates of Apriori
- ⇒ Has no infrequent subsets

Pass 1: Count F_S and N_S over DB

F may contain itemsets not in F_S or N_S
How to find them?

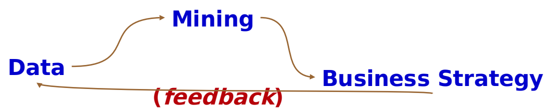
- Promoted borders: Itemsets in N_S that become frequent
- Observation 1: If there are any promoted borders, then our estimate of *F* is wrong
- Observation 2: Only supersets of the promoted borders can be frequent

Pass 2: Generate such supersets and find their counts over *DB*
- Do *not* generate itemsets which have infrequent subsets (i.e. N_S – {promoted borders})

Analysis of Sampling

- ✓ Estimate of frequent itemsets is usually quite accurate
  - ✓ Exact frequent itemsets not needed in many applications
- ✗ Sampling itself may require one database scan if random access is not possible (due to lack of index)
- ✗ If exact frequent itemsets are required, the cpu-cost is more than Apriori because
  - ✗ same counting technique as Apriori
  - ✗ first scan counts similar number of candidates as Apriori
  - ✗ cpu-cost of second scan is extra

Incremental Mining



- Database = *DB* ∪ *db*
  - *DB* – original database
  - *db* – increment
- Find rules for *DB* ∪ *db* and for *db*
- Input: F_{DB}, N_{DB}
- Output: $F_{DB \cup db}$, $N_{DB \cup db}$, F_{db}, N_{db}

Idea 1: Treat *DB* and *db* as two *partitions*. Apply partition technique.

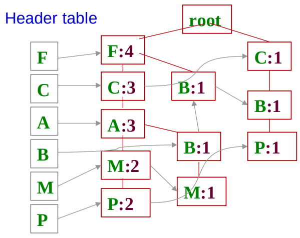
Idea 2: Treat *DB* as a *sample*. Apply sampling (negative border) technique.

Idea 3: Idea 1 + Idea 2

Practice Problem

Show the steps of Apriori on the following dataset with mincount = 3. Clearly show candidate and frequent itemsets of each length. Dataset: facdgimp, abcflmo, bfhjo, bcksp, afcelpmn.

FP-Growth: Growing the FP-Tree



Insert_FPtree(r, T):

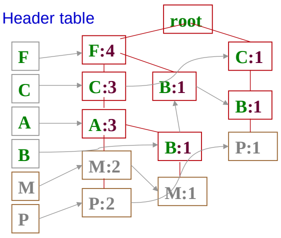
1. sort r by frequency, keep only frequent items
2. starting from root, follow path P as long as sequence of elements in P is a *prefix* of sorted record r
3. we reach a node n, cp to the longest prefix of r in T
4. add a path P' of nodes as descendents of n to hold remaining elements of r. Maintain linked lists from header table
5. last node in path $P+P'$ represents new transaction
6. increment counters of all nodes in $P+P'$

Dataset
FCAMP
FCABM
FB
CBP
FCAMP

Construct_Tree( ):
T = empty tree with root
for each transaction t:
  insert(r, T)

Preprocessing: *only* frequent items –- sorted by frequency

FP-Growth: Mining the FP-Tree



Conditional Pattern Base of B
FCA : 1
F : 1
C : 1

Call FP-Growth(T, null)

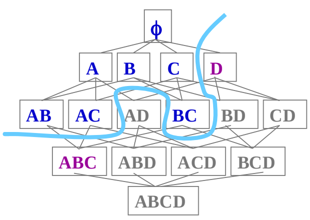
FP-Growth(T, α):
if T contains a single path P
  for each combination β of nodes in P
    generate pattern $\beta \cup \alpha$ with
      minimum support among nodes in β
else for each a_i in header table of T
  generate pattern $\beta = a_i \cup \alpha$ with a_i.support
  construct β's conditional pattern base
  construct β's conditional fp-tree T'
  if T' is non-empty
    FP-Growth(T', β)

Analysis of FP-Growth

- ✓ Reuses work done for processing transactions that share a common prefix
- ✓ Counting technique is thus different from Apriori
  - ✓ No explicit data-structure traversal for each transaction
- ✓ Very effective for *dense* datasets
- ✗ Not effective for *sparse* datasets
  - ✗ If a sequence of items appears in even one transaction, it will be represented in the fp-tree
  - ✗ For a large random sparse dataset, every sequence is likely to appear in at least one transaction
  - ✗ The fp-tree then grows linearly with database size
- ✗ Invalid claims of "no candidates"
  - ✗ Every candidate that is counted in Apriori is also counted in FP-growth (and a counter is maintained for it)

Problem Formulation

- Input
  - Database D
  - Minimum support threshold *minsup*
- Output:
  - *Frequent Itemsets F*: Itemsets whose *support* $\geq$ *minsup*
  - *Negative Border N: Minimally infrequent itemsets*
  - *Supports of* $F \cup N$



Feeding Frenzy

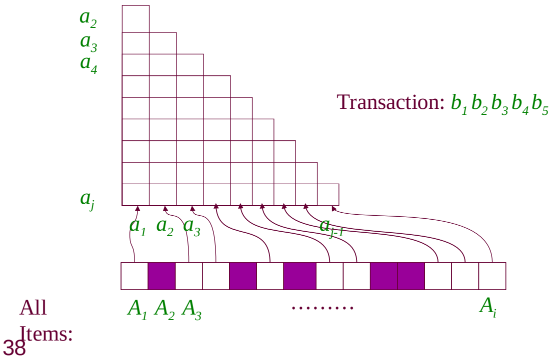
Apriori Sampling ASCPA FP-Growth
AIS Partition DIC CARMA VIPER ····· Is there an end? Optimal
1993 1995 1997 1999 2001

AIS	ClusterApr	SPINC
SETM	Eclat	AS-CPA
Apriori	MaxEclat	SSAS-CPA
AprioriTid	Clique	RSAS-CPA
AprioriHybrid	MaxClique	ColumnWise
OCD	TopDown	Hierarchical BitMap
Partition	dEclat	FP-Growth
Sampling	OPUS_AR	Tree Projection
DHP	BFPA	VIPER
CARMA	DFPA	H-Mine

Optimal Algorithm: Oracle

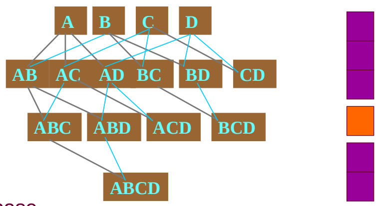
- Magically knows identities of frequent itemsets before mining begins. Therefore, has to only determine the counts of these itemsets in *one* pass over the database
- Minimum work required from any algorithm
- Careful design of data structures to ensure optimal access and enumeration of itemsets

Counting 1&2-itemsets



Counting Longer Itemsets (k >2)

DAG Structure

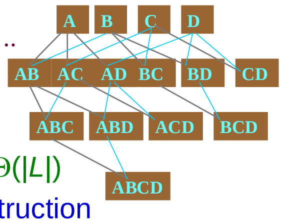


- Partition database
- For each itemset, compute the list of transaction-ids (*tidlist*) containing it
- Initiate tidlist intersections from frequent singletons
- Depth-first traversal
- Optimize using tid-vector approach

Tidset Intersection

Tid-vector V: 0010101101….

Tid-list L: 5, 6, 10, 29, …



- Cost of intersection = $\Theta(|L|)$
- Cost of tid-vector construction
  - Proportional to number of "1"s in V
  - Amortized over many intersections
  - Space for V can be statically allocated

No wasted Enumeration

- All 1-itemsets are either frequent or in -ve border
- Only combinations of *frequent* 1-itemsets enumerated for pairs
- Depth-first search ensures each itemset is visited only *once*

Enumeration Cost = $\Theta(1)$

- Direct lookup arrays for 1&2-itemsets. Best in unit-cost RAM model
- For longer itemsets, cost = $\Theta(|X.childset|)$ resulting in $\Theta(1)$ cost per itemset overall
- All operations involve array and pointer lookups, which cannot be improved upon

Oracle Features

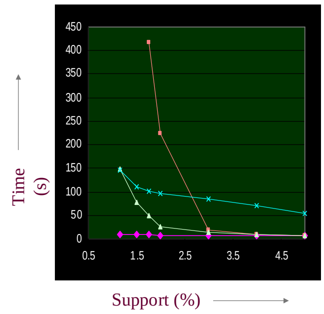
- Uses direct lookup arrays for 1-itemsets and 2-itemsets
- Uses DAG structure for longer itemsets
- No wasted enumeration of itemsets
- Enumeration cost per itemset = $\Theta(1)$
- Caveat: Not really optimal
  - Doesn't share work for transactions that are significantly similar. E.g. if 2 transactions are identical, it does the same work for both

Performance of Current Algorithms

Performance Setup

- Algorithms: Oracle, VIPER, FP-growth, Apriori
- Variety of Databases
  - File-system backend
  - Integration with commercial RDBMS
    - Cache data to file-system and run algorithm
    - Implement algorithm as stored procedure
    - Implement algorithm in SQL
- Extreme and typical values of *minsup*

Response Times of Current Algorithms



Oracle VIPER FP-Growth Apriori

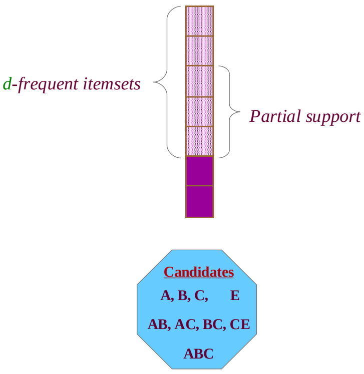
Online Algorithm

ARMOR: Association Rule Mining based on ORacle

ARMOR

- *Minimal* changes to Oracle
- Maximum *two* passes over database
- "Short and light" second pass
- Performance: Within *twice* of Oracle for a variety of real and synthetic databases

ARMOR Processing



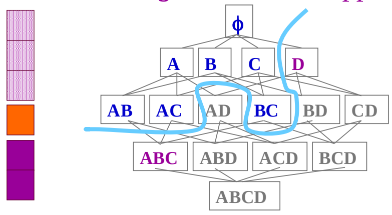
First Pass
- Conceptually *partition* database into disjoint blocks.
- *C* (the set of *candidates*) = set of all 1-itemsets
- After processing each partition, *update* *C*; this may involve both insertions and deletions
- *d-frequent itemsets*: The algorithm ensures that at any stage, if *d* is the database scanned so far, then the frequent itemsets within *d* are available.
- *Partial Supports*: The count of an itemset *since* it has been inserted into *C*.

Second Pass
- Obtain *complete supports* of candidates.
- No new insertions into *C*; but deletions allowed.
- Light and short pass

Candidate Generation

Itemsets can move freely between being partially-frequent, negative border and partially-infrequent.

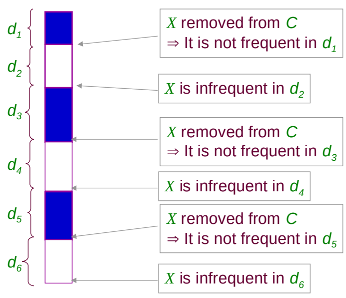
The Negative Border Approach



Observation: An itemset can become partially frequent iff it has some subset in *N* which moves to *F*. Such itemsets are called *promoted borders.*

Proof of Correctness

Consider the life of an itemset *X* in the set of candidates, *C*
Solid area represents that *X* was in *C*
Blank area represents that *X* was *not* in *C*

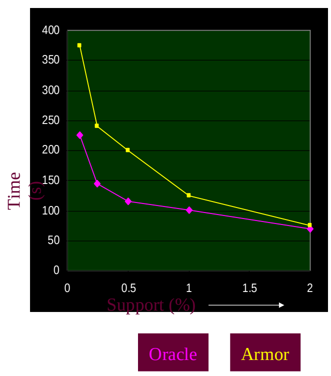


Since *X* is infrequent in every block, it is infrequent overall.

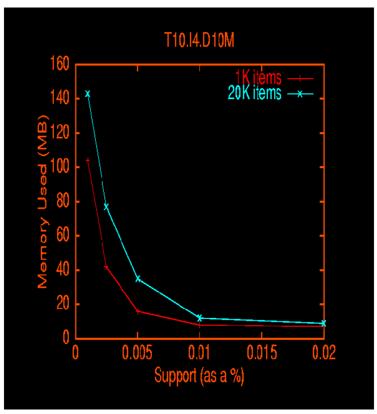
51

Performance of ARMOR

Response Times of ARMOR



Memory Utilization of ARMOR



54

Conclusions: **Is ARMOR++ Feasible?**

- Number of candidates in ARMOR are only 10% more than minimum (all frequent itemsets+negative border)
- Number of passes is effectively less than two
- So, scope for improvement appears to be limited …
- Caveat: Doesn't share work for transactions that are significantly similar. E.g. if 2 transactions are identical, it does the same work for both

Talk Outline

- Introduction
- First-time Mining of Association Rules
- Conciseness of Mining Results
  - Background
  - Closed Itemsets Framework
- Conclusions

Problem: Too many rules!

Dataset	*minsup*	#frequent
Sparse	0.1%	27,532
Dense	70%	48,969

Most are redundant

Post-mining Rule Pruning Schemes

- E.g. Output only rules that satisfy a user-given *minimum improvement*.
  - Improvement: Minimum difference between confidence of a rule and any of its sub-rules with the same RHS.
- Problem: What if mining itself is infeasible due to large output size?

Constrained Association Rules

- User specifies constraints on what kind of rules he is looking for. E.g. RHS should contain *milk.*
- Problem:
  - User may not have any constraints in mind.
  - Artificial, if purpose is just to reduce number of rules.

Maximal Frequent Itemsets

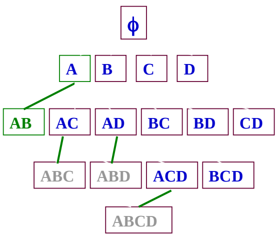
- A maximal frequent itemset is one that has no frequent supersets. (Also, called *positive border*.)
- Problems:
  - Identity of subsets can be deduced, but not their supports.
  - Subsets may have unexpected supports and thus be interesting on their own.
  - Cannot be used to form rules, since supports of subsets is necessary for this.

Closed Itemsets *[Zak00]*

Closed Itemsets Definition

- Tidset of an itemset X:
  - $t(X)$ = set of tids of transactions containing X
- Itemset of a tidset T:
  - $i(T)$ = items common to all transactions in T
- Closure Operator $c(X) = i(t(X))$:
  - Extension: $X \subseteq c(X)$
  - Monotonicity: If $X \subseteq Y$, then $c(X) \subseteq c(Y)$
  - Idempotency: $c(c(X)) = c(X)$
- Closed Itemset X: Iff $c(X) = X$

Closed Itemsets Redefinition



support(A) = support(AB)
support(AC) = support(ABC)
Etc.

An itemset is closed iff it has no superset with same support.

Mining Frequent Closed Itemsets

In any algorithm for frequent itemset mining:

- If an itemset has a subset with same support, don’t generate any of its supersets as candidates.
- E.g. in Apriori, remove such itemsets from F before applying AprioriGen(F)

Problem with Closed Set Approach

- Exact Support Equality: Requires supports of some itemsets and their supersets to be ***exactly*** equal.
- Mushroom Database Example: Addition of 438 tuples to 8,124 tuple database causes number of closed frequent itemsets at *minsup*=20% to increase from 1,390 to 15,541 – 11 times!

Talk Outline

- Introduction
- First-time Mining of Association Rules
- Conciseness of Mining Results
- Conclusions

Architecture for Association Rule mining