# Computing Association Rules

Dimitrios Gunopulos, UCR

Nikos Koudas, ATT Research

## Association Rules

- Given:
  - a database of transactions
  - each transaction is a set of items
- Find all association rules that satisfy user-specified minimum support and minimum confidence constraints.
- *Example:* 30% of transactions that contain beer also contain diapers; 5% of transactions contain these items
  - 30% : *confidence* of the rule
  - 5% : *support* of the rule
- We are interested in *finding* all rules rather than *verifying* if a rule holds.
- In this tutorial, we follow the presentation of the VLDB'94 paper "Fast Algorithms for Mining Association Rules" by R. Agrawal, R. Srikant.

## Application Examples

- Market Basket Analysis
  - "$* \Longrightarrow$ Maintenance Agreement"

    What the store should do to boost Maintenance Agreement sales?
  - "Home Electronics $\Longrightarrow *$"

    What other products should the store stock up on if the store has a sale on Home Electronics?
- Attached mailing in direct marketing
- Detecting "ping-pong"ing of patients
  - transaction: patient
  - item: doctor/clinic visited by a patient
  - support of a rule: number of common patients
- HIC Australia "success story"
  - associations between medical payment codes
  - saved $500,000 per year per state

## Problem Statement

- $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ : a set of literals, called items.
- Transaction $T$ : a set of items such that $T \subseteq \mathcal{I}$.
- Database $\mathcal{D}$ : a set of transactions.
- A transaction $T$ *contains* $X$, a set of some items in $\mathcal{I}$, if $X \subseteq T$.
- An *association rule* is an implication of the form $X \Longrightarrow Y$, where $X, Y \subset \mathcal{I}$
- The rule $X \Longrightarrow Y$ holds in the transaction set $\mathcal{D}$ with *confidence* $c$ if $c$% of transactions in $\mathcal{D}$ that contain $X$ also contain $Y$.
- The rule $X \Longrightarrow Y$ has *support* $s$ in the transaction set $\mathcal{D}$ if $s$% of transactions in $\mathcal{D}$ contain $X \cup Y$.

Find all rules that have support and confidence greater than user-specified minimum support and minimum confidence.

## Problem Decomposition

1. Find all sets of items that have minimum support (*frequent itemsets*).
2. Use the frequent itemsets to generate the desired rules.

## Problem Decomposition – Example

| Transaction ID | Items Bought |
|---|---|
| 1 | Shoes, Shirt, Jacket |
| 2 | Shoes, Jacket |
| 3 | Shoes, Jeans |
| 4 | Shirt, Sweatshirt |

For Minimum Support = 50% = 2 transactions, and Minimum Confidence = 50%:

| Frequent Itemset | Support |
|---|---|
| {Shoes} | 75% |
| {Shirt} | 50% |
| {Jacket} | 50% |
| {Shoes, Jacket} | 50% |

For the rule Shoes $\Longrightarrow$ Jacket :

- Support = Support({Shoes, Jacket}) = 50%
- Confidence $= \frac{\text{Support}(\{\text{Shoes, Jacket}\})}{\text{Support}(\{\text{Shoes}\})} = \frac{50}{75} =$ 66.6%.

Jacket $\Longrightarrow$ Shoes has 50% support and 100% confidence.

## Discovering Rules (1)

**Naive Algorithm**

**for each** frequent itemset $l$ **do**
  **for each** subset $c$ of $l$ **do**
    **if** (support($l$)/support($l - c$) $\geq$ *minconf*) **then**
      **output** the rule $(l - c) \Longrightarrow c$,
        with confidence = support($l$)/support($l - c$)
        and support = support($l$);

Example: $l = ABC$

| $c$ | $A$ | $B$ | $C$ |
|---|---|---|---|
| $\frac{\text{supp}(l)}{\text{supp}(l-c)}$ | $\frac{\lvert ABC \rvert}{\lvert BC \rvert}$ | $\frac{\lvert ABC \rvert}{\lvert AC \rvert}$ | $\frac{\lvert ABC \rvert}{\lvert AB \rvert}$ |
| Rule | $BC \Longrightarrow A$ | $AC \Longrightarrow B$ | $AB \Longrightarrow C$ |

| $c$ | $AB$ | $BC$ | $AC$ |
|---|---|---|---|
| $\frac{\text{supp}(l)}{\text{supp}(l-c)}$ | $\frac{\lvert ABC \rvert}{\lvert C \rvert}$ | $\frac{\lvert ABC \rvert}{\lvert A \rvert}$ | $\frac{\lvert ABC \rvert}{\lvert B \rvert}$ |
| Rule | $C \Longrightarrow AB$ | $A \Longrightarrow BC$ | $B \Longrightarrow AC$ |

## Discovering Rules (2)

**Lemma**. If a consequent $c$ generates a valid rule, so do all subsets of $c$. (e.g. $X \Longrightarrow YZ$, then $\exists\ XY \Longrightarrow Z$ and $XZ \Longrightarrow Y$.)

Consider the rule $(l - c) \Longrightarrow c$.

Now, if $c_1 \subset c$,

$$
\begin{aligned}
l - c_1 &\supset l - c \\
\text{support}(l - c_1) &\leq \text{support}(l - c) \\
\text{support}(l)/\text{support}(l - c_1) &\geq \text{support}(l)/\text{support}(l - c) \\
\text{conf}((l - c_1) \Longrightarrow c_1) &\geq \text{conf}((l - c) \Longrightarrow c)
\end{aligned}
$$

**Example** Consider a frequent itemset $ABCDE$.

If $ACDE \Longrightarrow B$ and $ABCE \Longrightarrow D$ are the only one-consequent rules with minimum confidence, then
$ACE \Longrightarrow BD$ is the only other rule that needs be tested.

## The Apriori Algorithm

- $L_k$ : Set of frequent itemsets of size $k$ (those with minimum support).
- $C_k$ : Set of candidate itemsets of size $k$ (potentially frequent itemsets)

$L_1 = \{$frequent items$\}$;
**for** ( $k = 1$; $L_k \neq \emptyset$; $k{+}{+}$ ) **do**
    **begin**
    $C_{k+1}$ = New candidates generated from $L_k$;
    **foreach** transaction $t$ in the database **do**
      Increment the count of all candidates in $C_{k+1}$ that
          are contained in $t$.
    $L_{k+1}$ = Candidates in $C_{k+1}$ with minimum support.
    **end**
Answer = $\bigcup_k L_k$;

# Apriori – Example

Database $\mathcal{D}$

| TID | Items |
|---|---|
| 10 | 1 3 4 |
| 20 | 2 3 5 |
| 30 | 1 2 3 5 |
| 40 | 2 5 |

Minimum Support = 50% = 2 trans.

Scan $\mathcal{D}$ $\longrightarrow$

$C_1$

| Itemset | Sup. |
|---|---|
| {1} | 2 |
| {2} | 3 |
| {3} | 3 |
| {4} | 1 |
| {5} | 3 |

$L_1$

| Itemset | Sup. |
|---|---|
| {1} | 2 |
| {2} | 3 |
| {3} | 3 |
| {5} | 3 |

$C_2$

| Itemset |
|---|
| {1 2} |
| {1 3} |
| {1 5} |
| {2 3} |
| {2 5} |
| {3 5} |

Scan $\mathcal{D}$ $\longrightarrow$

$C_2$

| Itemset | Sup. |
|---|---|
| {1 2} | 1 |
| {1 3} | 2 |
| {1 5} | 1 |
| {2 3} | 2 |
| {2 5} | 3 |
| {3 5} | 2 |

$L_2$

| Itemset | Sup. |
|---|---|
| {1 3} | 2 |
| {2 3} | 2 |
| {2 5} | 3 |
| {3 5} | 2 |

$C_3$

| Itemset |
|---|
| {2 3 5} |

Scan $\mathcal{D}$ $\longrightarrow$

$C_3$

| Itemset | Sup. |
|---|---|
| {2 3 5} | 2 |

$L_3$

| Itemset | Sup. |
|---|---|
| {2 3 5} | 2 |

## Apriori Candidate Generation

*Monotonicity Property*: All subsets of a frequent itemset are frequent.

Given $L_k$, generate $C_{k+1}$ in two steps:

$$L_3 = \boxed{\begin{array}{c} \{1\ 2\ 3\} \\ \{1\ 2\ 4\} \\ \{1\ 3\ 4\} \\ \{1\ 3\ 5\} \\ \{2\ 3\ 4\} \end{array}}$$

1. *Join* Step : Join $L_k$ with $L_k$, with the join condition that the first $k-1$ items should be the same **and** $l^1[k] < l^2[k]$.

   Now, $C_4 = \{\{1\ 2\ 3\ 4\},\ \{1\ \underline{3\ 4\ 5}\}\ \}$.

2. *Prune* Step : Delete all candidates which have a non-frequent subset.

   Now, $C_4 = \{\{1\ 2\ 3\ 4\}\}$.

## Apriori Candidate Generation (2)

Why does this maintain completeness?

- *Monotonicity Property*: All subsets of a frequent itemset are frequent.
- The join step is equivalent to extending each itemset in $L_k$ with every item in the database and then deleting those itemsets $c$ whose subset $(c-c[k])$ is not frequent.

## Candidate Generation: Example

| | | |
|---|---|---|
| 1 2 3 | ~~1~~ | Earlier in the |
| | ~~2~~ | lexicographic |
| | ~~3~~ | ordering |
| | 4 | |
| | 5 | |
| | ~~6~~ | 6 not frequent |
| 1 2 4 | 5 | |
| | ~~6~~ | 6 not frequent |
| 1 3 4 | 5 | |
| | ~~6~~ | 6 not frequent |
| 1 3 5 | 6 | 6 not frequent |
| 2 3 4 | 5 | |
| | ~~6~~ | 6 not frequent |

| | |
|---|---|
| 1 2 3 4 | |
| 1 $\boxed{3\ 4\ 5}$ | {3 4 5} not frequent |

1 2 3 4

## Finding Candidates Contained in a Transaction

Given

- a transaction $T$ and
- a set of candidates $C_k$,

find all members of $C_k$ which are contained in $T$.

$C_2$ : { {1, 2}, {1, 3}, {3, 4}, {5, 6} }
$T$ : {3, 4, 6}

Only check itemsets in buckets corresponding to 3, 4, and 6, i.e., {3,4}.

- avg. number of items in trans. $\ll$ total number of items
- generalized into a *hash-tree*

## Improving the Apriori

- Sampling
- More efficient candidate generation
- Find the closed itemsets only
- Find the maximal frequent itemsets only

## A discovery model

Given a database $\mathbf{r}$,

a language $\mathcal{L}$ for defining groups of the data, and

an interestingness criterion $q$ for evaluating if a sentence $\phi \in \mathcal{L}$ is interesting,

Find $Th(\mathcal{L}, \mathbf{r}, q) = \{\phi \in \mathcal{L} \mid q(\mathbf{r}, \phi) = TRUE\}$.

## A discovery model

In addition, we assume:

- A monotone specialization relation $\preceq$:

  $\preceq$ is a partial order on the sentences in $\mathcal{L}$.

  If $\psi \preceq \phi$ and $q(\mathbf{r}, \phi) = TRUE$, then $q(\mathbf{r}, \psi) = TRUE$

- The sentences are represented as itemsets:

  there exists a set $R$ of items (attributes) and a function $f : \mathcal{L} \to \mathcal{P}(R)$ such that if $\psi \preceq \phi$ then $f(\psi) \subseteq f(\phi)$.

Given $\mathcal{L}$, $\mathbf{r}$, $q$ and $\preceq$, compute $MTh(\mathcal{L}, \mathbf{r}, q, \preceq)$.

## $\sigma$-**frequent itemsets**

Given a binary relation $\mathbf{r}$ and a support value $0 < \sigma < 1$, a $\sigma$-**frequent itemset** $F$ is a itemset of columns (attributes) for which at least a $\sigma$ fraction of the rows have value 1.

Relation

| | A | B | C | D |
|---|---|---|---|---|
| $T_1$ | 1 | 0 | 1 | 0 |
| $T_2$ | 1 | 1 | 1 | 0 |
| $T_3$ | 1 | 0 | 1 | 1 |
| $T_4$ | 0 | 0 | 0 | 1 |
| $T_5$ | 0 | 1 | 0 | 1 |
| $T_6$ | 0 | 0 | 1 | 1 |
| $T_7$ | 1 | 1 | 1 | 1 |
| $T_8$ | 0 | 1 | 1 | 0 |

0.5-freq itemset

| A | C |
|---|---|
| 1 | 1 |
| 1 | 1 |
| 1 | 1 |
| 0 | 0 |
| 0 | 0 |
| 0 | 1 |
| 1 | 1 |
| 0 | 0 |

.25-freq itemset

| B | D |
|---|---|
| 0 | 0 |
| 1 | 0 |
| 0 | 1 |
| 0 | 1 |
| 1 | 1 |
| 0 | 1 |
| 1 | 1 |
| 1 | 0 |

Let $Fr(\mathbf{r}, \sigma)$ be the set of $\sigma$-frequent itemsets and $MFr(\mathbf{r}, \sigma)$ be the set of maximal $\sigma$-frequent itemsets.

**The Problem:**

Given a binary relation **r**, and a support value $\sigma \in [0, 1]$, find all the maximal $\sigma$-frequent itemsets in **r**, $MFr(\mathbf{r}, \sigma)$.

## Computing Maximal $\sigma$-frequent itemsets

- Given a 0/1 relation $r$, and a threshold $\sigma$, the problem of finding the number of $\sigma$-frequent itemsets is #P-complete.

  (Reduction from the MAX-2SAT problem.)

- Given a 0/1 relation $r$, and a threshold $\sigma$, the problem of deciding if there is a maximal $\sigma$-frequent itemset with at least $T$ attributes is NP-complete.

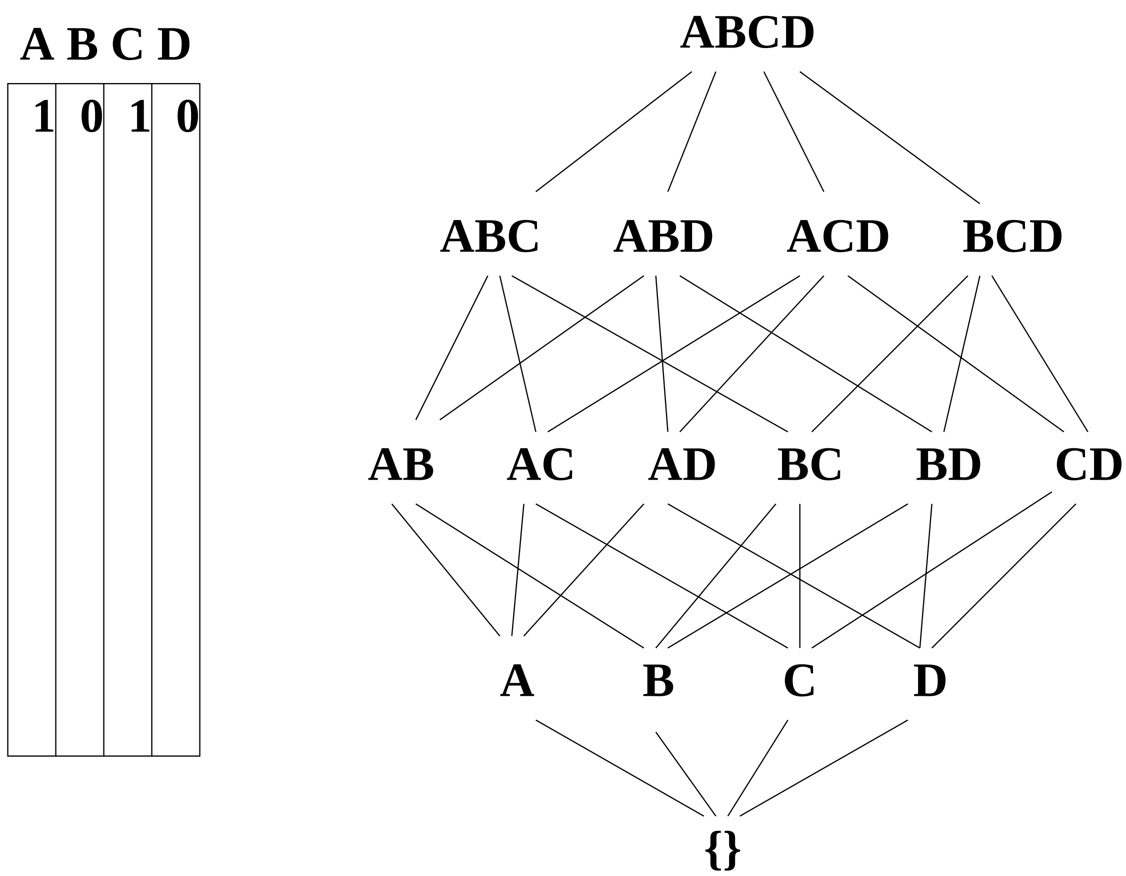
  (Reduction from the $K_{N,N}$ clique in bipartite graph problem.)

Complexity measures:

Output complexity.

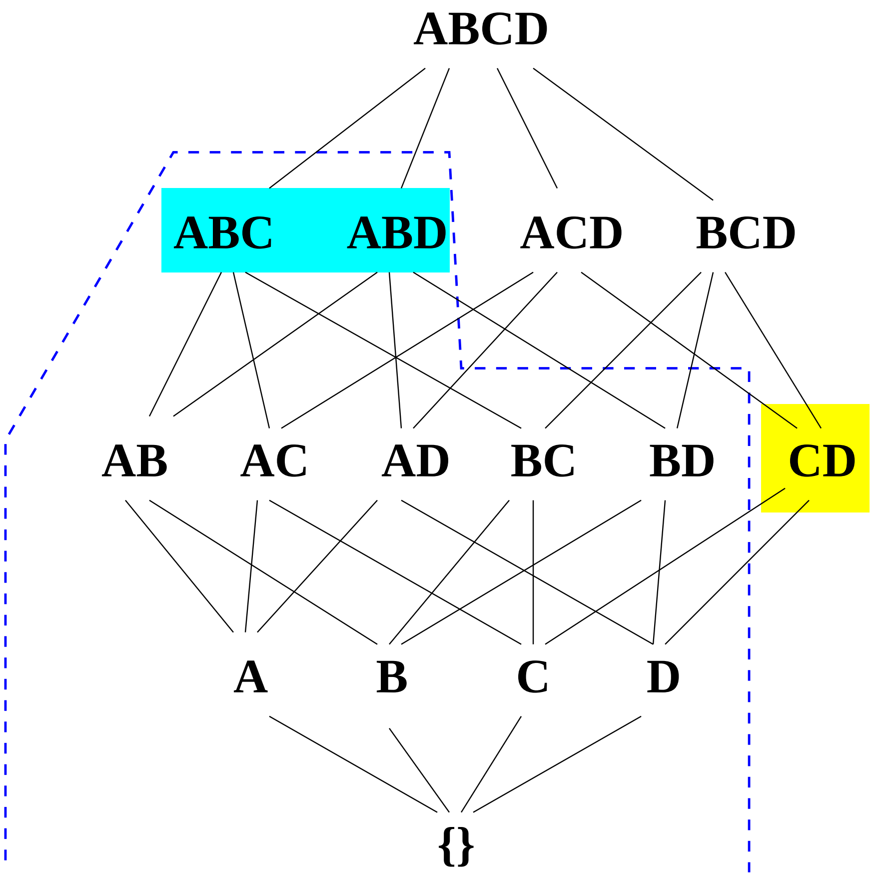
Number of Is-freq queries.

# Computing Maximal $\sigma$-frequent itemsets




The *lattice* and the 0/1 relation matrix

# A lower bound

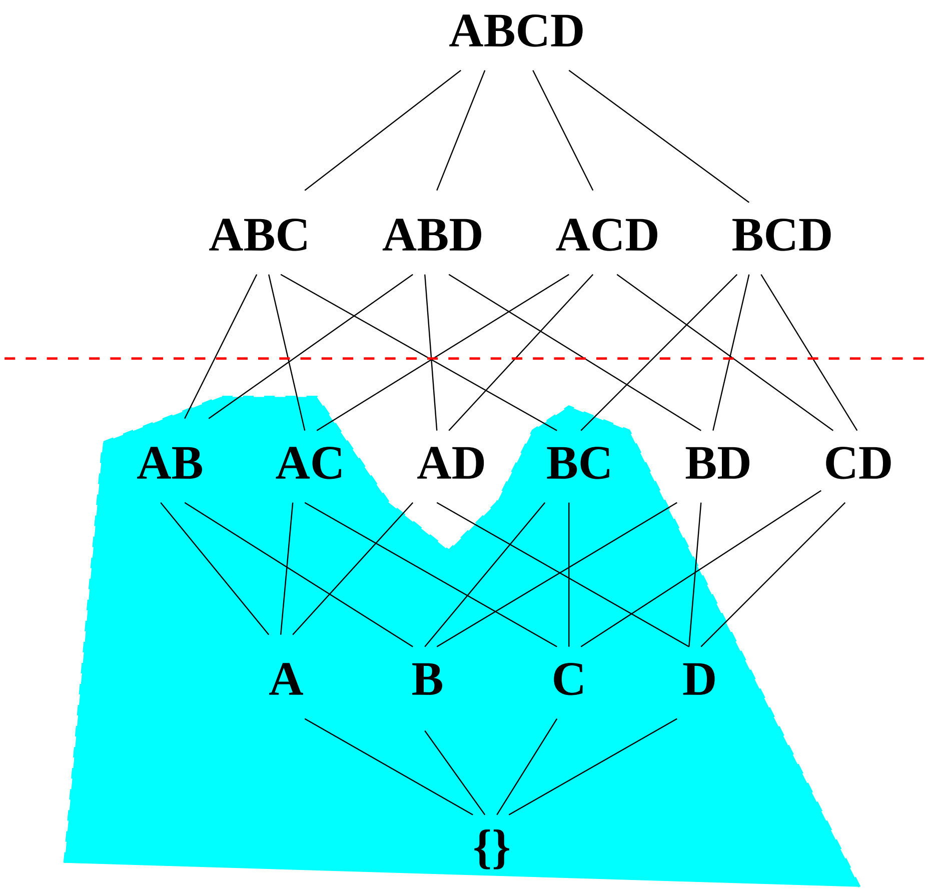



Let $Bd^-$ be the minimal not $\sigma$-frequent itemsets, and

$$Bd = MFr(\mathbf{r}, \sigma) \cup Bd^-.$$

Computing $MFr(\mathbf{r}, \sigma)$ requires at least $|Bd|$ **Is-freq** queries.

## The *apriori* algorithm



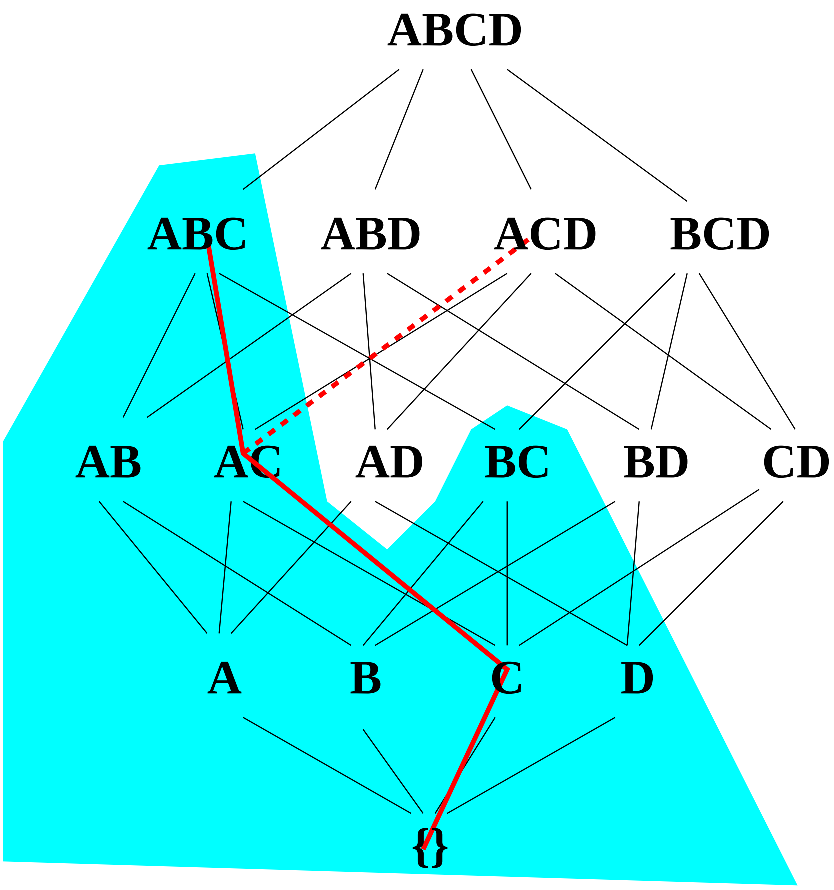

1. Find the set $F_1$ of $\sigma$-frequent itemsets in level 1.

2. From $F_i$, compute the set $C_{i+1}$ of possible $\sigma$-frequent itemsets in level $i+1$

3. Find which itemsets from $C_{i+1}$ are $\sigma$-frequent.

# Find one maximal $\sigma$-freq itemset

Take a random ordering $p$ of the attributes. Compute the lexicographically first, w.r.t. $p$, maximal $\sigma$-freq itemset (in $O(nm)$ time or $m$ Is-freq queries)




For $p$=(C,A,D,B) this is {A,B,C}.

## The randomized algorithm

- Algorithm Find_MFS:

  Given a relation **r** with attributes $R_1, \ldots, R_{|R|}$ and a threshold $\sigma$, find a maximal $\sigma$-frequent itemset.

  1. Find a random permutation $p$ of $(1, \ldots, |R|)$:

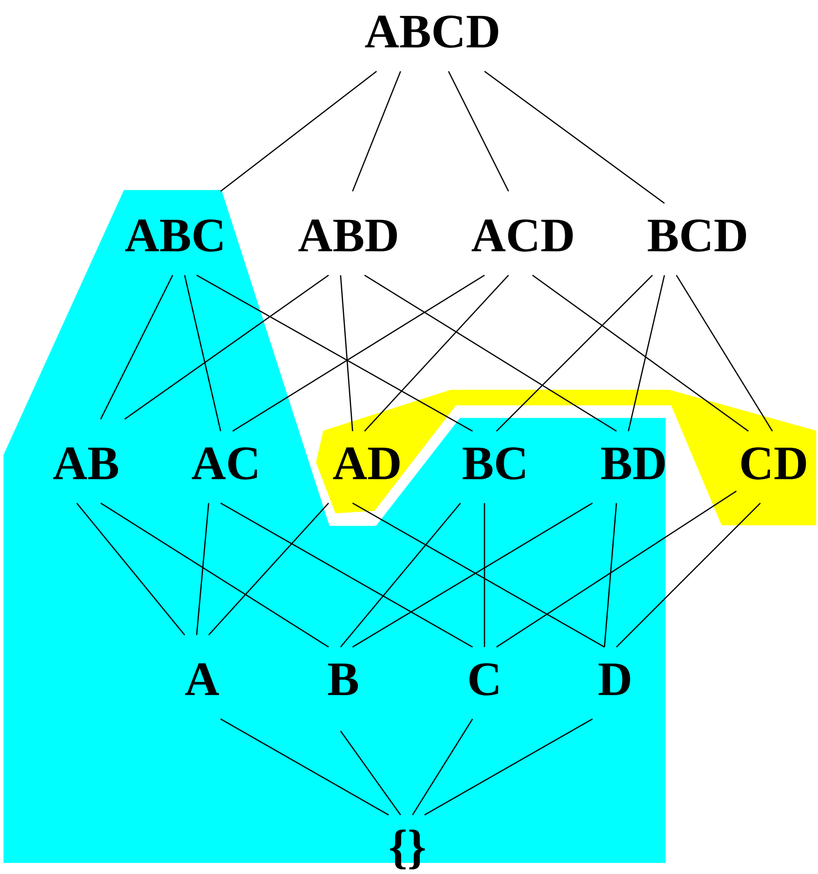
  2. Set $MFS = \{\}$.

  3. For $i = 1$ to $|R|$:

     (a) If $MFS \cup \{R_{p(i)}\}$ is a $\sigma$-frequent itemset, add $R_{p(i)}$ to $MFS$.

  4. Return $MFS$

## Computing $MFr(\mathbf{r}, \sigma)$




Maximal $\sigma$-frequent itemsets found so far: $\{A, B, C\}, \{B, D\}$.

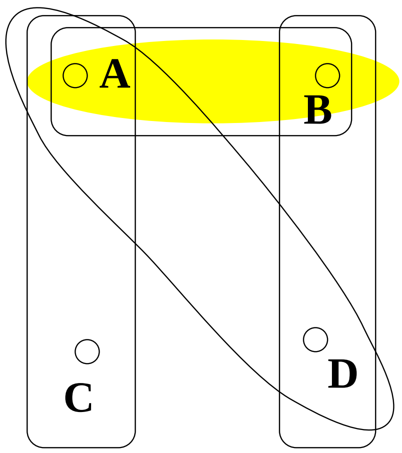
A new maximal $\sigma$-frequent itemset must include a itemset on the negative boundary ($\{A, D\}$ or $\{C, D\}$).

## Transversals

A *hypergraph* is a 0-1 matrix with distinct rows. Each row is called a *hyperedge* and each column is called an *element*.

A *transversal* of a hypergraph is a set of elements such that every hyperedge contains at least one element from the set (every row has an 1 in at least one of the columns in the itemset.)

| | A | B | C | D |
|---|---|---|---|---|
| $E_1$ | 1 | 0 | 1 | 0 |
| $E_2$ | 1 | 0 | 0 | 1 |
| $E_3$ | 1 | 1 | 0 | 0 |
| $E_4$ | 0 | 1 | 1 | 0 |




Transversal: {A,B}

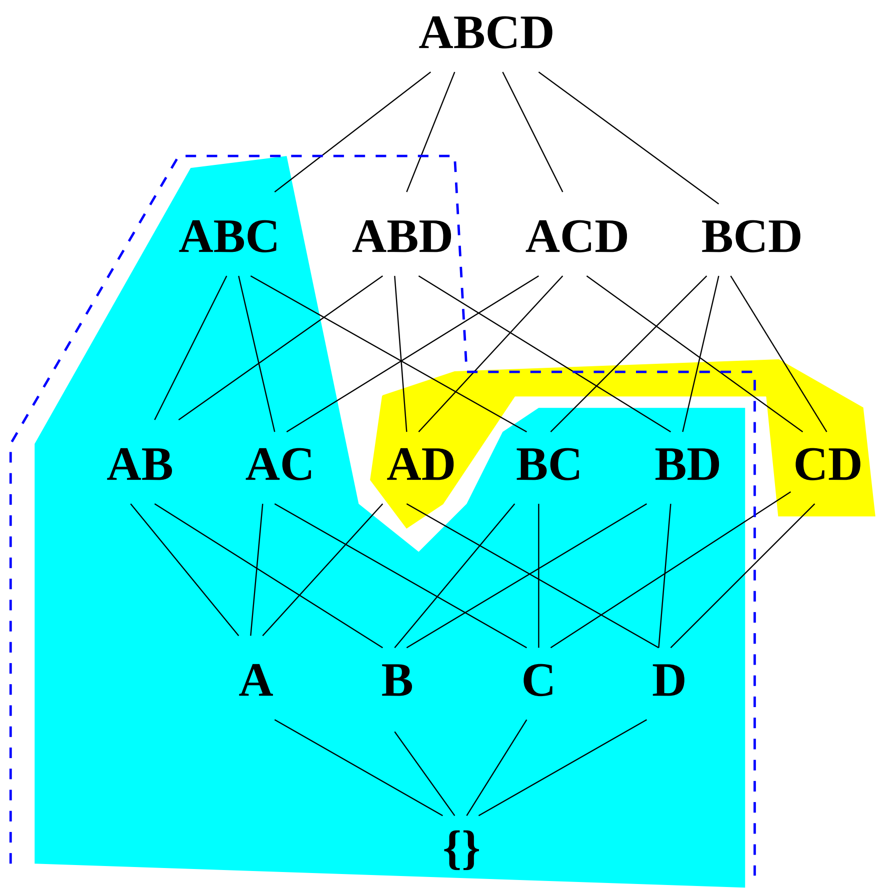
## Transversals and maximal $\sigma$-frequent itemsets

### Theorem

Given a set S of maximal frequent itemsets of a given matrix and support value, if there exists a maximal frequent itemset outside S, then the new frequent itemset *must contain a minimal transversal* of the hypergraph of complements of frequent itemsets in S.

Enumeration of all minimal transversals of a hypergraph with *polynomial delay* is a well known problem.

# Transversals and maximal $\sigma$-frequent itemsets




Each traversal either belongs to $Bd^-$ or to $Fr(\mathbf{r}, \sigma)$.

## Transversals and maximal $\sigma$-frequent itemsets

The following problems are equivalent:

- Computing maximal $\sigma$-frequent itemsets using only Is-freq queries.

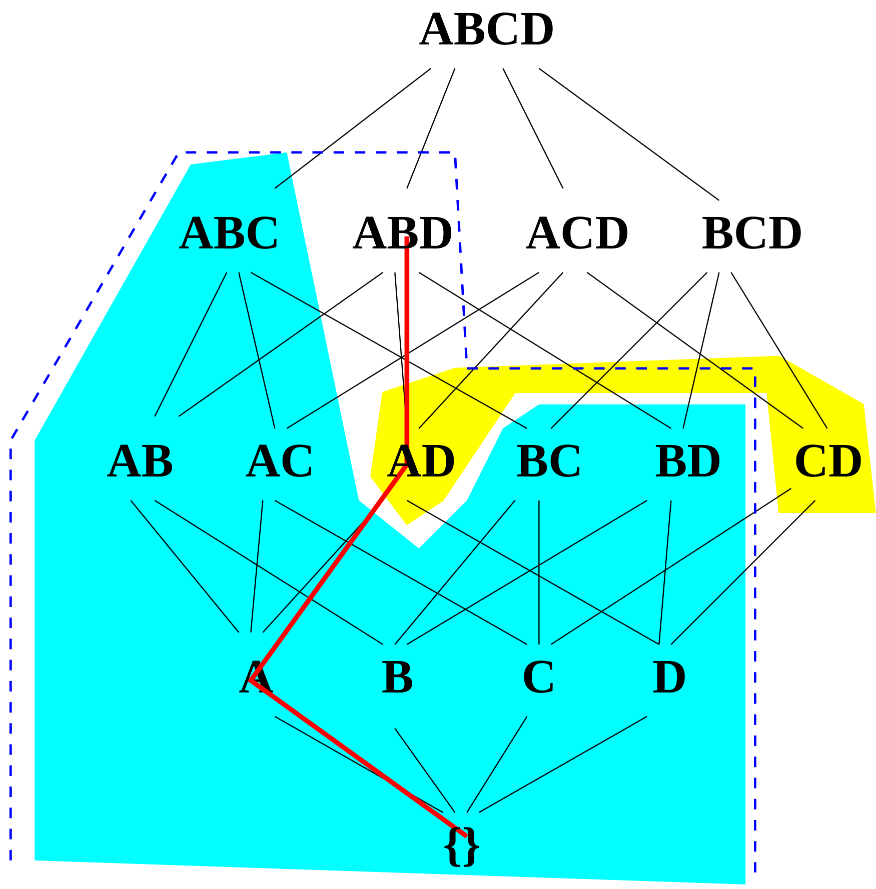
- Computing hypergraph transversals using only Is-transversal queries.

  Set $q(\mathbf{r}, \theta) = TRUE$ iff $\theta$ is not a transversal.

- Learning monotone boolean functions using only Membership queries.

  Set $q(\mathbf{r}, \theta) = TRUE$ iff setting the variables in $\theta$ to 1 is a satisfying assignment.

## Using the transversals




From the $\sigma$-frequent itemsets $\{A, B, C\}$ and $\{B, D\}$, find the hypergraph of the complements:

$$V = \{A, B, C, D\}, E = \{\{A, C\}, \{D\}\}$$

Its transversals are $\{A, D\}, \{C, D\}$.
$\{A, D\}$ is used as a "seed" for the ordering $\{A, D, B, C\}$.

## Computing all minimal transversals

- Algorithm Find_Trans:

  Given a hypergraph $G = (V, E)$,

  1. Compute the *lookahead matrix* for $G$.

  2. Run a backtracking algorithm using the lookahead matrix to output all transversals.

**The Dualize and Advance algorithm:**

Given a relation $\mathbf{r}$ with attributes $R_1, \ldots, R_{|R|}$ and a threshold $\sigma$, compute $MFr(\mathbf{r}, \sigma)$.

1. Set $MFr(\mathbf{r}, \sigma) = \{\}$, $TR = \{\}$.

2. While new itemsets are added in $MFr(\mathbf{r}, \sigma)$ do:

   (a) Find a transversal $TR = Tran(G_{R, \overline{M}Fr(\mathbf{r},\sigma)})$ that is not in $|Bd^-|$.

   (b) Run algorithm Find_MFS on $\mathbf{r}$, $TR$.

   (c) Add the result to $MFr(\mathbf{r}, \sigma)$

3. Return $MFr(\mathbf{r}, \sigma)$.

**Theorem**

The problem of computing $MFr(\mathbf{r}, \sigma)$ using only Is-freq queries can be solved with $O(m|MFr(\mathbf{r}, \sigma)| + |Bd^-|)$ Is-freq queries and $O(|MFr(\mathbf{r}, \sigma)|)$ calls to a subroutine for enumerating hypergraph transversals.

- In each iteration, a new maximal $\sigma$-frequent itemset is discovered.
- Algorithm Find_MFS makes $O(m)$ Is-freq queries.
- $Tran$ is a transversal enumeration subroutine. Any set of $|Bd^-|+1$ transversals must include a frequent itemset.
- We store the itemsets in $Bd^-$.

Using a recent result by Fredman and Khachiyan this gives an $O(|Bd|^{\log|Bd|})$ running time for the problem.

They give an incremental algorithm to compute the transversals of a hypergraph in $T(I, i) = (I+i)^{O(\log(I+i))}$.

The algorithm works for any $\mathcal{L}$ that can be represented as itemsets.

## A generalized result

*Given* a database **r**,
a language $\mathcal{L}$,
and a frequency criterion $q$.

*Let* $Th(\mathcal{L}, \mathbf{r}, q) = \{\phi \in \mathcal{L} \mid q(\mathbf{r}, \phi) = true\}$

*Assume* that there is a partial order $\preceq$ , such that $q$ is monotone w.r.t. $\preceq$
(for all $\psi, \theta \in \mathcal{L}$, if $\psi \preceq \theta$ and $q(\mathbf{r}, \psi)$ then $q(\mathbf{r}, \theta)$)

**Theorem**

The Dualize and Advance algorithm computes all the most specific sentences in $Th(\mathcal{L}, \mathbf{r}, q)$.

## Summary

- The Apriori algorithm is very efficient when the size of the maximal frequent itemsets is small.

- The Dualize and Advance and similar algorithms are more efficient if the frequent itemsets are large.

- The results hold for the more general case.

- Better strategies to handle sparse matrices efficiently are needed.

- Lower bounds for a more realistic model.

- Use similar techniques for more complicated problems (sequential patterns).