CSCI 5408 Data Analytics: DM and DW Tech (Week 10)

- Ass5 Due: 28
  - Ass5-Tutorial: Mar 15
- Write answers for review questions
  - Final Exam: Apr 20, 3:30-5:30 PM
- Reading: Lecture 15-16; Text: Ch1, Ch6 of 3rd edition (or Ch5 of 2nd edition)

Application e.g. of AR Mining: Market Basket Analysis

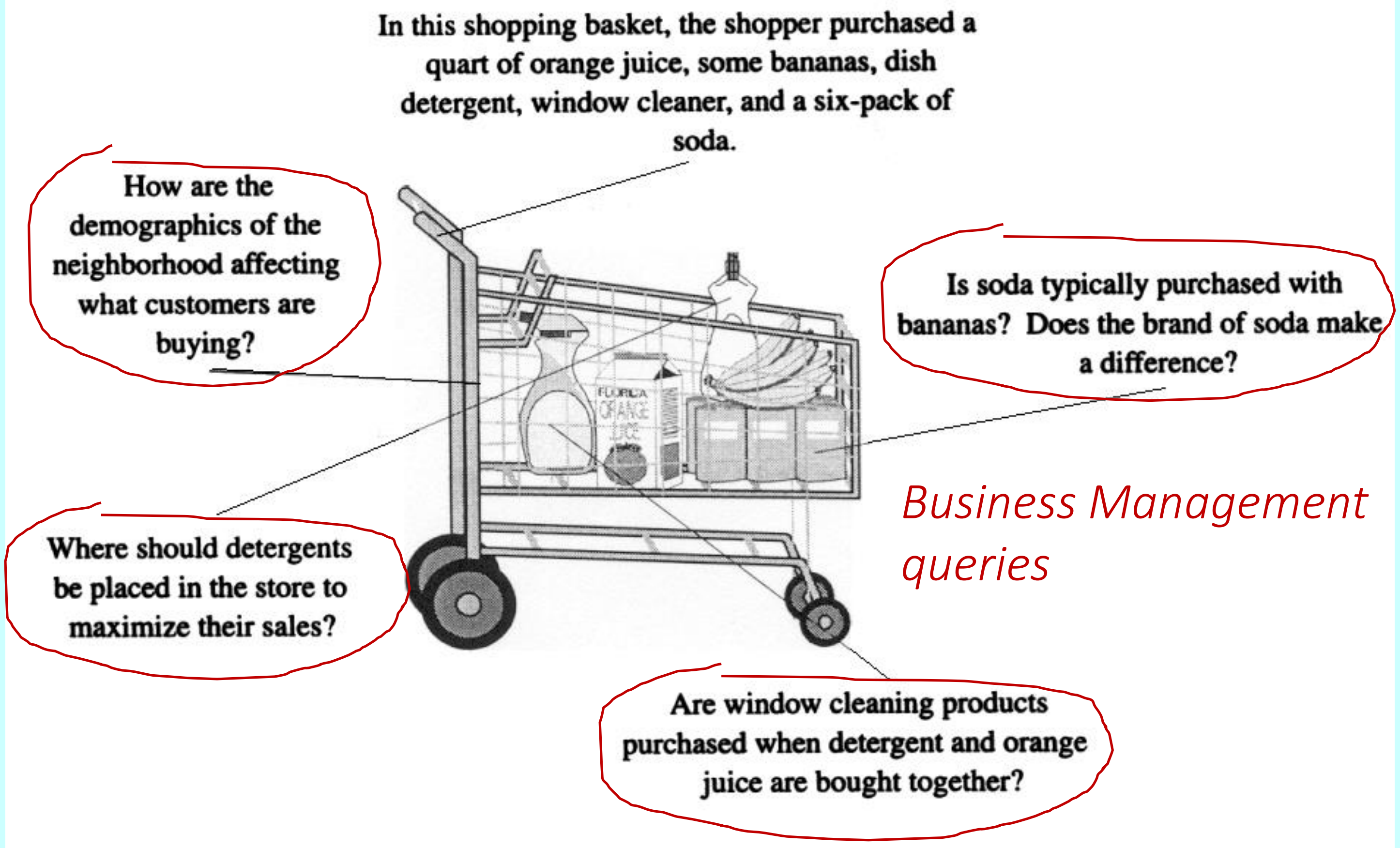
- **Motivation:**

The generation of **association patterns about the business data objects** produces intuitively obvious results that the business owner can use in order to take advantage of the knowledge within their industry to improve their business.

- **Market basket analysis:**
  - **To find interesting relationships among retail products**
    - To analyze customer buying pattern (habits/trends) by finding associations among different items that customers place in their "shopping baskets".
  - **All possible combinations of potentially interesting product groupings can be explored**
    - "Which items are likely to be purchased together for a customer on a given trip to the store?"
    - One basket can tell about one customer's one trip shopping, but all purchases made by all customers have much more information.
  - **Business improvements:** design promotions, arrange shelf or catalog items, develop cross-marketing strategies, CRM, etc.

Market Basket Analysis (Cont)

Association Rule: $\boldsymbol{X} \Rightarrow \boldsymbol{Y}$ *(sup, conf)*



Business Management queries

Customer transaction pattern analysis

Analyze grocery sale transactions:

TID	Items
1	orange juice, soda
2	orange juice, milk, window cleaner
3	orange juice, detergent
4	orange juice, detergent, soda
5	soda, window cleaner

The association rule:

When a customer buys orange juice he/she will also buy soda:

orange juice $\Rightarrow$ soda [40%, 50%]

Support: 40% transactions have both items.

Confidence: 50% transactions containing orange juice also contain soda.

Transactional Data Concepts

- Database: all transactions, $D = \{t_1, t_2, \ldots, t_n\}$
  - E.g., $D = \{t_{10}, t_{20}, t_{30}, t_{40}\}$
- Set of items: all unique items in D, $I = \{I_1, I_2, \ldots, I_m\}$
  - E.g., $I = \{A,B,C,D,E,F\}$
- Itemset: a set of items, $t_i = \{I_{i1}, I_{i2}, \ldots, I_{ik}\}$, $I_{ij} \subseteq I$
  - E.g., $t_{10} = \{A,B,C\}$, $t_{20} = \{A,C\}$, $t_{30} = \{A,D\}$, $t_{40} = \{B,E,F\}$
  - E.g., {A}, {B}, …, {A,B}, {A,C}, …, {A,B,C},… {A,B,E,F}, …,{A,B,C,D,E,F}

Transaction-id	Items bought
10	A, B, C
20	A, C
30	A, D
40	B, E, F

Association Rule Definition

- **Association rule definition:**
  - Given a set of items I, and a DB of transactions D, an **association rule** is an implication of the form $X \Rightarrow Y$, where the two itemsets X and Y satisfy the condition: $X,Y \subseteq I$ and $X \cap Y = \emptyset$.
  - An association rule is considered as an item relationship pattern only if it passed the measures of minimum support and confidence rates.
- **Support rate of $X \Rightarrow Y$:**
  - The percentage of transactions in D containing both X and Y.
  - Support rate, *sup*, is the probability that a transaction contains both X and Y, i.e. $sup = P(X \cup Y)$.
- **Confidence rate of $X \Rightarrow Y$:**
  - The percentage of transactions containing X also contain Y.
  - Confidence rate, *conf,* conditional probability that a transaction having X also contains Y, i.e. $conf = P(Y|X)$.

Support and Confidence Rates (cont)

- Support Rate (*Sup*) of $X \Rightarrow Y$:
  - If n is the number of transactions in dataset *D, the number of transactions* containing both *X and Y* is denoted by $(X \cup Y)$.count, then

$$\boxed{sup = \frac{(X \cup Y).count}{n}}$$

- Confidence Rate (*conf*) of $X \Rightarrow Y$:
  - If the number of transactions containing *X* only is denoted by *X.count*, then

$$\boxed{conf = \frac{(X \cup Y).count}{X.count}}$$

Association Rule Definition (cont)

- Following the original definition (by Agrawal et al., 93), the problem of association rule mining is defined as:
  - Let ***I*** = $\{i_1, i_2, \ldots i_n\}$ be a set of binary attributes called items.
  - Let ***D*** = $\{t_1, t_2, \ldots t_n\}$ be a set of transactions called the database.
  - Each transaction in ***D*** has a unique transaction ID and contains a subset of the items in ***I***.
- A *rule* is defined as an implication of the form:
  $X \Rightarrow Y$, *where* $X, Y \subseteq \mathbf{I}$.
  - Every rule is composed by two different sets of items, also known as itemsets, *X* and *Y*, where *X* is called ***antecedent*** or left-hand-side (LHS) and *Y* ***consequent*** or right-hand-side (RHS).

Organize transactional data with table

- Given a transactional dataset in supermarket domain:

Example database with 5 transactions and 5 items					
transaction ID	milk	bread	butter	beer	diapers
1	1	1	0	0	0
2	0	0	1	0	0
3	0	0	0	1	1
4	1	1	1	0	0
5	0	1	0	0	0

- The set of unique items is *I = {milk, bread, butter, beer, diapers}* and the shows a small table containing the items, where, in each entry, the value 1 means the presence of the item in the corresponding transaction, and the value 0 represents the absence of an item in that transaction.
  - An example rule for the supermarket could be {butter, bread} => {milk} meaning that if butter and bread are bought, customers also buy milk, with measures of sup = 20%, conf = 100%.

(*Note: this example is extremely small. In practical applications, a rule needs a support of several hundred transactions before it can be considered statistically significant[, and datasets often contain thousands or millions of transactions.)

Association Rule Mining Approach

(A two-phase process algorithm)

- Two major procedures:
  1. **Find frequent itemsets**
     - Search all frequent itemsets from the DB
  2. **Generate association rules**
     - Derive rules from each frequent itemset

- Which procedure needs more computational effort to deal with & why?

Illustration: find frequent itemsets

Transaction-id	Items bought
1	A, B, C
2	A, C
3	A, D
4	B, E, F

- Given a transaction dataset, D
- The set of unique items of D:
  I = {A, B, C, D, E, F}, it tells how many one item itemsets are contained in the DB: {A}, {B}, {C}, {D}, {E}, {F}.
- Other itemsets:
  - Each itemset is one of possible combinations of items in I.
  - Each transaction t is just an itemset instance of I.
  - How many of them?

Find frequent itemsets: How many?

- **How many itemsets can be generated from I?**
- E.g., For I={A,B,C,D,E,F}, the following itemsets can be generated which may, or may not appear in D:

1-item itemsets: {A}, {B}, {C}, {D}, {E}, {F}

2-item itemsets: {A,B}, {A,C}, …

3-item itemsets: {A,B,C}, …

4-item itemsets:{A,B,C,D}, …

5-item itemsets:{A,B,C,D, E}, …

6-item itemsets:{A,B,C,D,E,F}

Transaction-id	Items bought
1	A, B, C
2	A, C
3	A, D
4	B, E, F

Search Space: How many itemsets need to be searched?

D:

Transaction-id	Items bought
10	A, B, C
20	A, C
30	A, D
40	B

- I = {A, B, C, D}

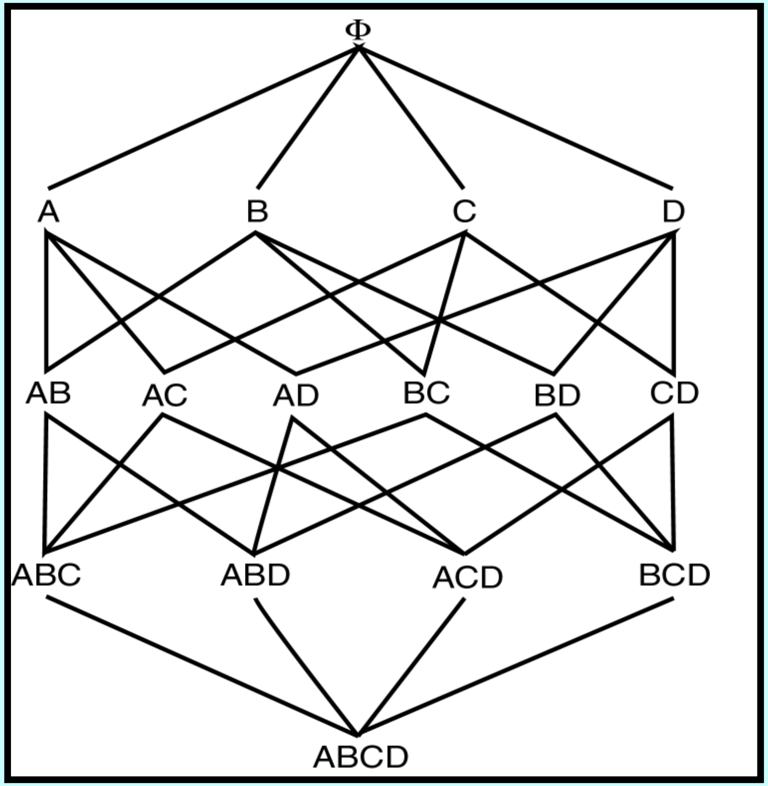
- The unique items

- Search space:

- All unique itemsets: 15

Search space:

A lattice of itemsets for I = {A, B, C, D}



Search Space

If a database has n unique items, and k denotes the combinations of k items ($k \le n$), the possible combinations:

$$\sum^{N}_{k=1} C^K, \quad \text{where } C^K = n! / ((n-k)! * k!)$$

E.g., when n = 4, the sum of

k=1,	4!/((4-1)!*1!) = 4	A,B,C,D
k=2,	4!/((4-2)!*2!) = 6	AB,AC,AD,BC,BD,CD
k=3,	4!/((4-3)!*3!) = 4	ABC,ABD,ACD,BCD
k=4,	4!/((4-4)!*4!) = 1	ABCD

$\sum^{4}_{k=1} C^4$, i.e. the number of itemsets = $\mathbf{2^4 - 1 =}$ 15.

When n = 100, the number of itemsets = ?

$2^{100} = 1.27 \times 10^{30} - 1$

How big is it?

How many itemsets **actually occurred in D (vs. the Lattice)**?

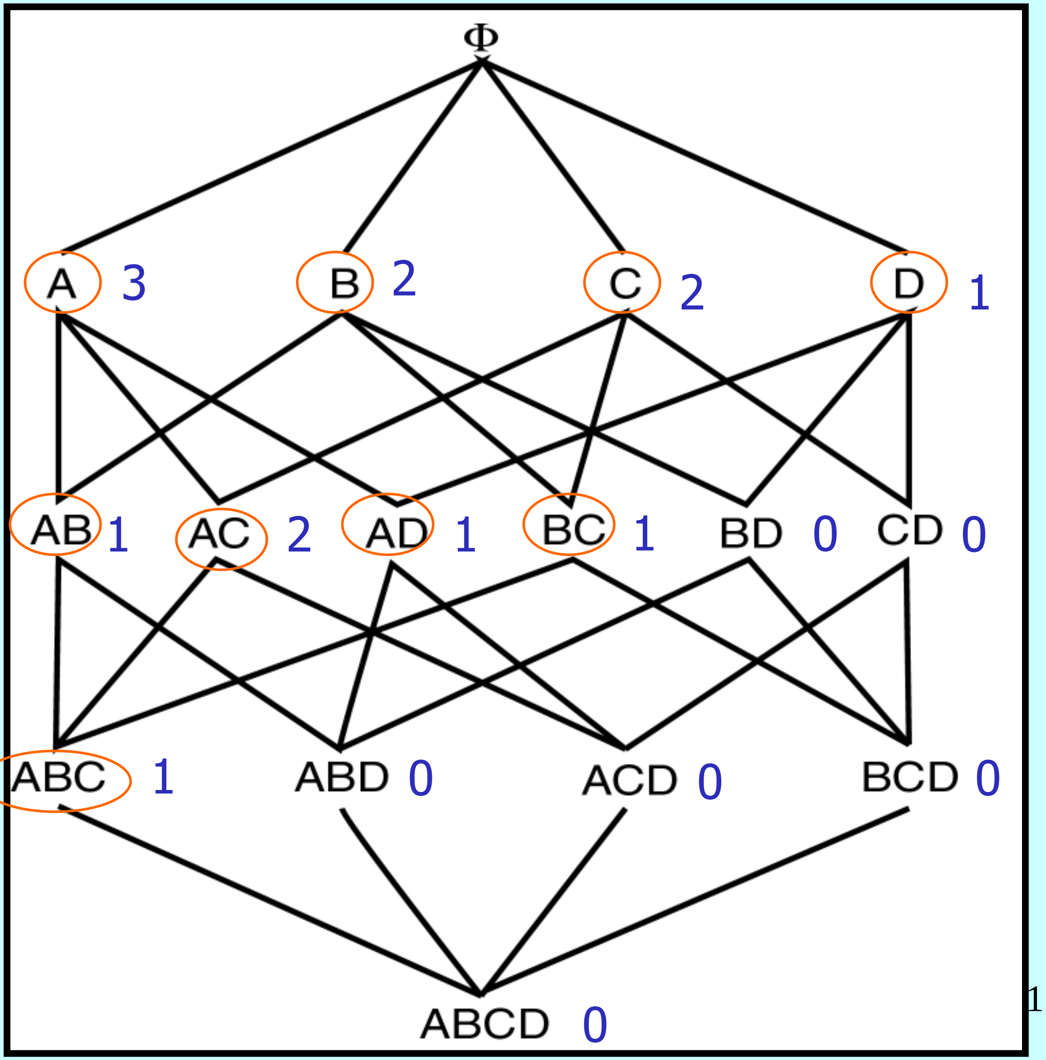
D:

Transaction-id	Items bought
10	A, B, C
20	A, C
30	A, D
40	B

What is the ground truth?
* The unique itemsets contained in D are only a subset of the lattice (with various counts).

Do we need to search whole space, and why?

Lattice of itemsets for I = {A, B, C, D}



How many itemsets need to be searched?

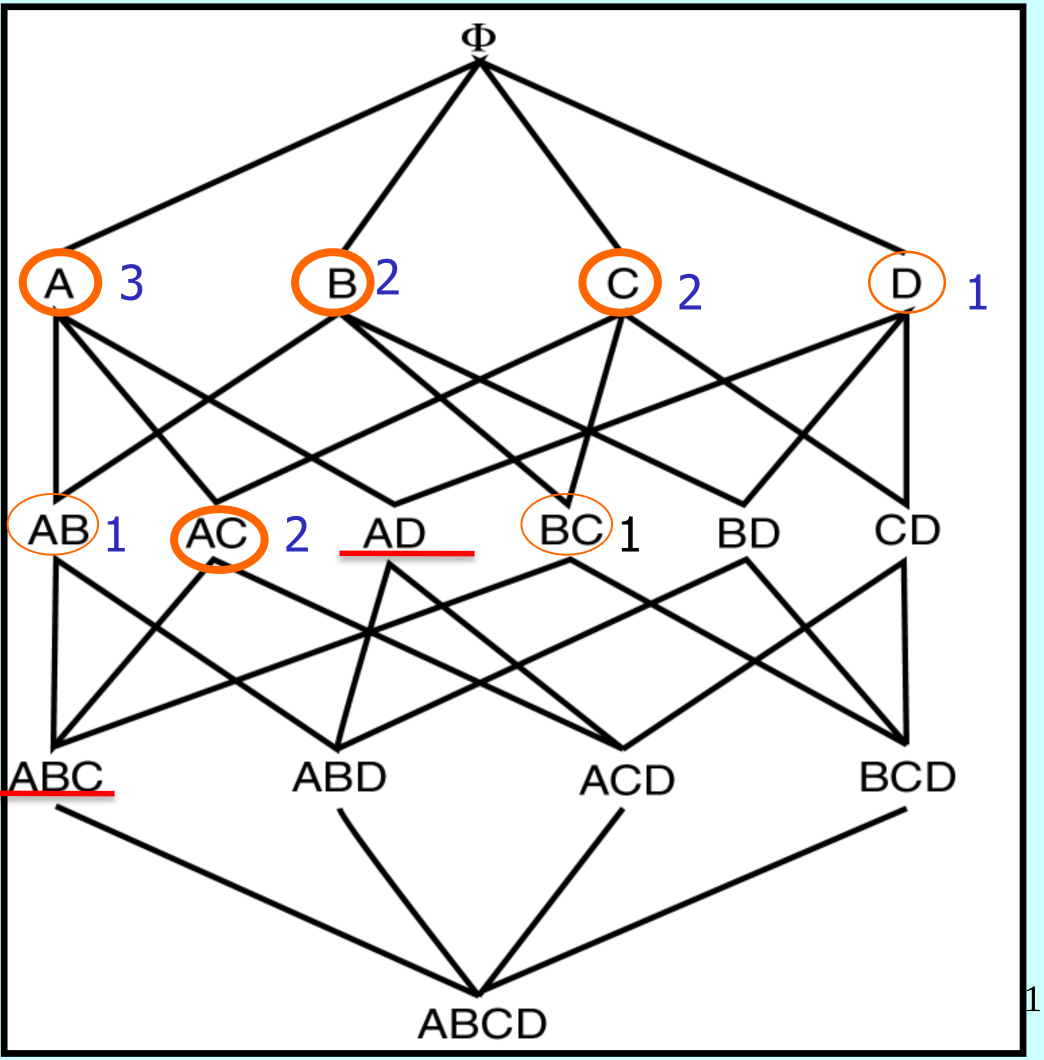
D:

Transaction-id	Items bought
10	A, B, C
20	A, C
30	A, D
40	B

For I = {A,B,C,D}, the search space: 15 itemsets.

- If the given sup = 50%, i.e. count=2, how may itemsets need to be searched, why?

Lattice of itemsets for I = {A, B, C, D}



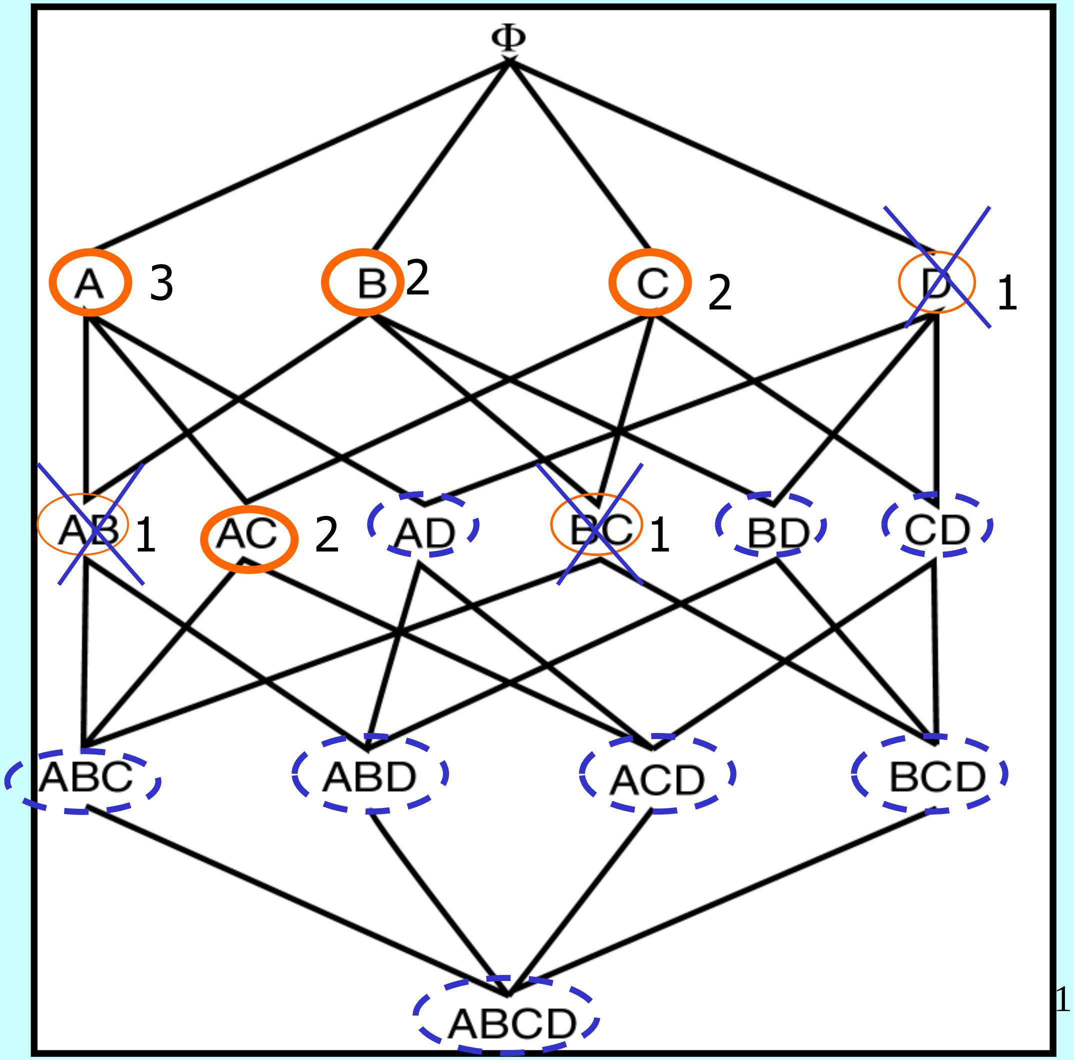
Search Pruning

D:

Transaction-id	Items bought
10	A, B, C
20	A, C
30	A, D
40	B

Heuristic: If a itemset is not frequent, all of its supersets are not frequent, i.e. **no need to search them (i.e. generate them).**

Lattice of itemsets for I = {A, B, C, D}



If an itemset is not frequent, none of its supersets is frequent.

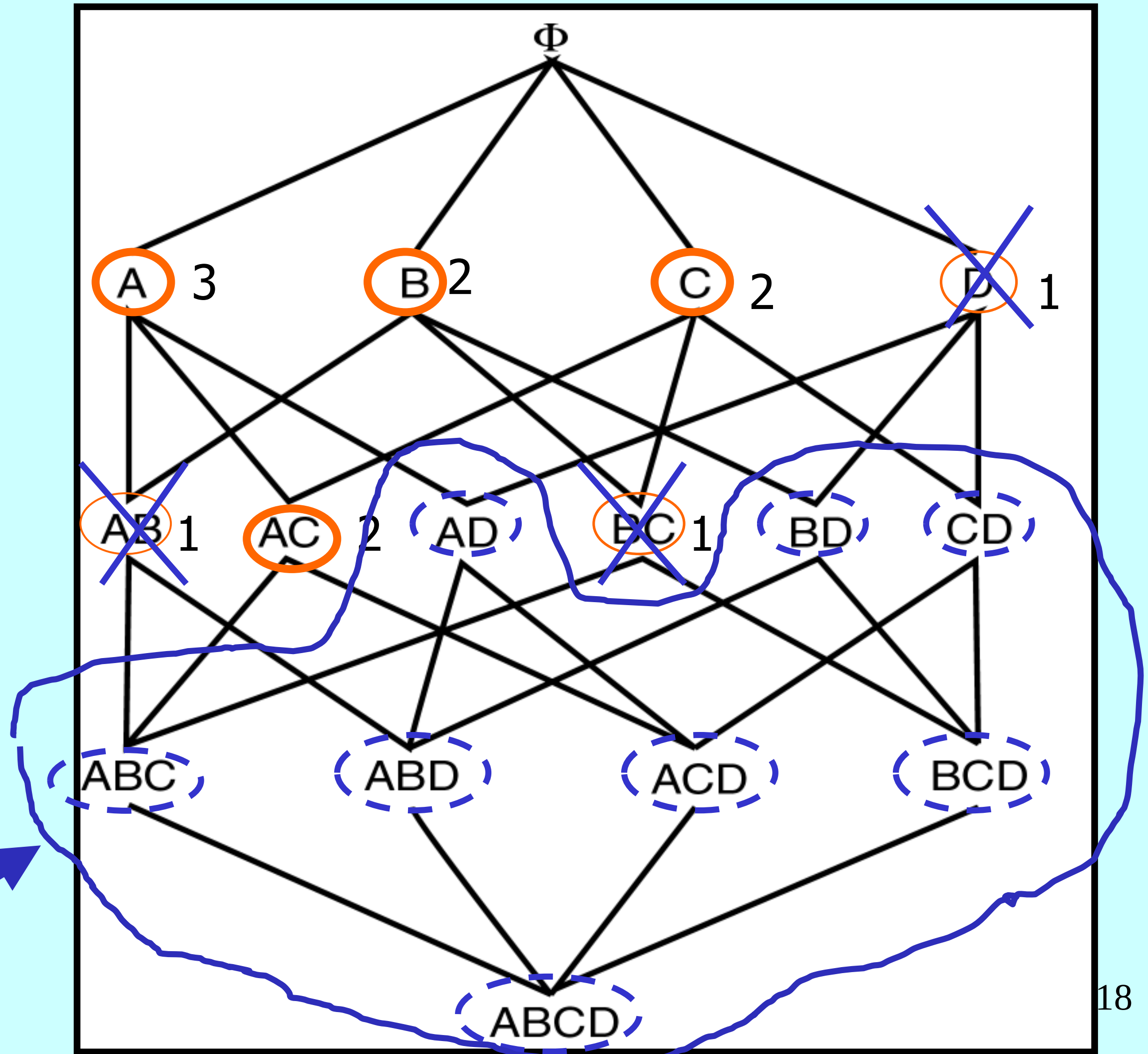
D:

Transaction-id	Items bought
10	A, B, C
20	A, C
30	A, D
40	B

Heuristic: If a itemset is not frequent, all of its supersets are not frequent, i.e. **no need to search them** (i.e. no need to generate them).

Pruned search space

Lattice of itemsets for I = {A,B,C,D}



A priori knowledge: Apriori Property

- Frequent Itemset Property:

  Any subset of a frequent itemset is frequent.

- Contra-positive:

  If an itemset is not frequent, none of its supersets is frequent.

How can this knowledge be applied in designing a more efficient algorithm?

AR Mining: Search space pruning

- Two major procedures:
  1. Find frequent itemsets
  2. Generate rules from frequent itemsets
- **How to make the computation more efficient?**
  - Do we need to search/generate the entire space, i.e. generate all itemsets, why?
  - What itemsets need to be searched?
  - Strategy: Apply "apriori property" to prune the search space. How?

Apriori Algorithm: A Candidate Generation-and-Test Approach

- **Apriori pruning principle:** If there is any itemset which is infrequent, its superset <u>should not be generated for testing!</u>
- **Method:**
  - Generate length (k+1) candidate itemsets from length k <u>frequent itemsets only</u>, and
  - Test each (k+1) candidate to see if it passed Sup_rate by counting against D. Only qualified candidates are retained for next iteration.
- <u>**Any subset of a frequent itemset must be frequent**</u>
  - If {A, B, C} is frequent, so is any its subsets, such as {A,B},{A,C}, etc, since every transaction having {A, B, C} also contains {A,B},{A,C}, {B,C}, {A},{B},{C}. In other word, if an infrequent subset of {A, B, C} was found, then {A, B, C} would be qualified as a candidate itemset.
- The performance studies show its great efficiency and scalability:
  - Agrawal & Srikant 1994, Mannila, et al. 1994.

Association Rule Mining

- Two major procedures:
  1. **Find frequent itemsets**
  2. **Generate rules from frequent itemsets**
- **Issue:** How to make the computation more efficient?
  - Do we need to search (or generate) the entire space (i.e. generate all itemsets)?
  - What itemsets need to be searched?
  - **Strategy: Apply apriori property to prune the search space.**

Tracing Apriori Algorithm: given a TDB & Sup≥50%

Database TDB

Tid	Items
10	A, C, D
20	B, C, E
30	A, B, C, E
40	B, E

1st scan

C_1

Itemset	sup
{A}	2
{B}	3
{C}	3
{D}	1
{E}	3

L_1

Itemset	sup
{A}	2
{B}	3
{C}	3
{E}	3

C_2

Itemset
{A, B}
{A, C}
{A, E}
{B, C}
{B, E}
{C, E}

2nd scan

C_2

Itemset	sup
{A, B}	1
{A, C}	2
{A, E}	1
{B, C}	2
{B, E}	3
{C, E}	2

L_2

Itemset	sup
{A, C}	2
{B, C}	2
{B, E}	3
{C, E}	2

C_3

Itemset
{B, C, E}

3rd scan

L_3

Itemset	sup
{B, C, E}	2

Apriori Algorithm: A Candidate Generation-and-Test Approach

- **Key steps:**
  - Initially, scan DB once to get frequent 1-itemsets
  - **Generate** (k+1) **candidate** itemsets from k frequent itemsets
  - **Test the candidates (pruning)**
  - Terminate when no frequent or candidate set can be generated

Apriori Algorithm (Pseudo-code)

Input: Dataset D, min_sup
Output: L // frequent itemsets in D

C_k : Candidate k-itemsets;
L_k : Frequent k-itemsets;

```
L_1 = frequent_1-itemsets(D, min_sup);
for (k = 2; L_{k-1} !=∅; k++) {            // search in order
    C_k = apripri_gen(L_{k-1});            // candidates generation
    for each transaction t ∈ D {           // scan D for counts (candidate testing)
        C_t = subset(C_k , t);             // candidate subsets
        for each candidate c ∈ C_t
        c.count++;                 }
    L_k = {c∈ C_k | c.count ≥ min_sup}
return ∪_k L_k ;
```

Candidate Generation: $C_k = L_{k-1} \bowtie L_{k-1}$

apriori_gen(L_{k-1}, min_sup)

```
for each itemset l1 in Lk-1 {
    for each itemset  l2 in Lk-1
      if ( (l1[1] = l2[1])  ∧  (l1[2] = l2[2]) ∧  … ∧  (l1[k-1] = l2[k-1]) ∧
           (l1[k] < l2[k]) )  {
           c =  l1 ⋈ l2;              // join for generating candidates
           if has_infrequent_subset(c, Lk-1) {
             delete c;                 // prune unfruitful candidates
             else add c to Ck; }
      }
   }
}
return Ck;
```

Candidate Generation: **apriori_gen(L_{k-1})**

- Join Operation: $C_k = L_{k-1} \bowtie L_{k-1}$

- E.g., $k = 2$ and L_1 = {{1},{2},{3},{4}}

$$\boxed{\boldsymbol{C_2 = L_1 \bowtie L_1 = ?}}$$

1. $L_1 \times L_1$ = {{1},{2},{3},{4}} × {{1},{2},{3},{4}}
= ?

2. Remove redundant and irrelevant itemsets.

Generate (k+1)-itemsets by JOIN k-itemsets list with itself

E.g. Generate 2-itemsetes of the 1-temsets:
{{1},{2},{3},{4}}

$L_1 \times L_1$ = {{1},{2},{3},{4}} × {{1},{2},{3},{4}}
= ?

{1,1},{1,2},{1,3},{1,4},
{2,1},{2,2},{2,3},{2,4},
{3,1},{3,2},{3,3},{3,4},
{4,1},{4,2},{4,3},{4,4}

Analysis on JOIN Operation

$L_1 \times L_1$ = {{1},{2},{3},{4}} × {{1},{2},{3},{4}}
= ?

{1,1},{1,2},{1,3},{1,4},
{2,1},{2,2},{2,3},{2,4},
{3,1},{3,2},{3,3},{3,4},
{4,1},{4,2},{4,3},{4,4}

Redundant

Irrelevant

= {{1,2},{1,3}{1,4},{2,3},{2,4},{3,4}}

How to avoid to generate redundant and irrelevant combinations?

- Design a proper **Test Condition**

Review: JOIN Operation

- JOIN operation is very important for any relational database with more than a single relation (i.e. table), because it allows us to process relationships among relations.
- JOIN operation, denoted by $\bowtie$, is used to combine tuples from two tables into a single table in which new tuples are generated by cross product.
- Most common type of join is a "natural JOIN" (often just called "JOIN"). $R \bowtie S$ conceptually is:
  1. Compute $R \times S$.
  2. Select rows where attributes that appear in both relations have equal values.
  3. Project all unique attributes and one copy of each of the common ones.

Candidate Generation: apriori_gen(L_{k-1})

- The Join Operation: $C_k = L_{k-1} \bowtie L_{k-1}$
- E.g., $k = 4$ and L_3 = {{1 2 3},{1 2 4},{1 3 4},{1 3 5},{2 3 4}}

$$\boxed{C_4 = L_3 \bowtie L_3}$$

= {{<u>1 2 3</u>}, {1 2 4}, {1 3 4}, {1 3 5}, {2 3 4}}

$\bowtie$

{{1 2 3}, {1 2 4}, {1 3 4}, {1 3 5}, {2 3 4}} =

{1 2 3} $\bowtie$ {1 2 3} = {123123} = {123} X // Irrelevant
{1 2 3} $\bowtie$ {1 2 4} = {123124} = {1234} √
{1 2 3} $\bowtie$ {1 3 4} = {123134} = {1234} X // Redundant

...

Tracing JOIN process for generating C_k

$C_4 = L_3 \bowtie L_3$

$= \{\{1\ 2\ 3\}, \{1\ 2\ 4\}, \{1\ 3\ 4\}, \{1\ 3\ 5\}, \{2\ 3\ 4\}\} \bowtie$
$\{\{1\ 2\ 3\}, \{1\ 2\ 4\}, \{1\ 3\ 4\}, \{1\ 3\ 5\}, \{2\ 3\ 4\}\}$

$\{1\ 2\ 3\} \bowtie L_3 = \{1\ 2\ 3\ 1\ 2\ 4\} = \{1\ 2\ 3\ 4\}$
$\{1\ 2\ 4\} \bowtie L_3 = \{\text{empty}\}$ *// Not new 4-itemset is generated*
$\{1\ 3\ 4\} \bowtie L_3 = \{1\ 3\ 4\ 1\ 3\ 5\} = \{1\ 3\ 4\ 5\}$
$\{1\ 3\ 5\} \bowtie L_3 = \{\text{empty}\}$ *// Not new 4-itemset is generated*
$\{2\ 3\ 4\} \bowtie L_3 = \{\text{empty}\}$ *// Not new 4-itemset is generated*

After the join step, $C_4 = \{\{1\ 2\ 3\ 4\}, \{1\ 3\ 4\ 5\}\}$.

How to avoid to generate redundant and irrelevant combinations? - Design proper **Test condition**

Review Questions

1. How to estimate a search space given n unique items?
2. What is the Apriori knowledge/property, and why it is significant for the algorithm?
3. How is the Apriori property applied for pruning the search space? Trace the Apriori algorithm to identify the places where the Apriori knowledge is applied, and how?
4. For generating candidates of (k+1)-itemsets based on frequent k-itemsets, how to avoid generating irrelevant and redundant itemsets?