# Analytics and Visualization of Big Data

*Fadel M. Megahed*

## *Lecture 09: Association Rules and Similarity*

AUBURN UNIVERSITY
SAMUEL GINN
COLLEGE OF ENGINEERING

*Department of Industrial and Systems Engineering*

*Spring 13*

# Compacting the Output: An Exercise

- **Maximal Frequent itemsets**: no immediate superset is frequent

- **Closed itemsets:** no immediate superset has the same count (count>0).

| Itemset | Count | Maximal | Closed |
|---|---|---|---|
| A | 4 | No | No |
| B | 5 | No | Yes |
| C | 3 | No | No |
| AB | 4 | Yes | Yes |
| AC | 2 | Yes | No |
| BC | 3 | Yes | Yes |

For the above table, identify whether each of the following itemsets is frequent, maximal and/or closed. Use s≥ 3 to decide on whether an item is frequent or not

# Outline for Topics Covered in Chapter 06



**What is the Market-Basket Model?**

- Definition of Frequent Itemsets
- Applications of Frequent Itemsets
- Association Rules
- Finding Association Rules with High Confidence

**Market-Baskets and A-Priori Alg.**

- Representation of Market-Basket Data
- Main Memory for Itemset Counting
- Monotoncity of Itemsets
- Tyranny of Counting Pairs
- The A-Priori Algorithm and for All Frequent Itemsets

# Main-Memory Bottlenecks



- In many algorithms to find frequent itemsets we need to worry about how **main memory** is used.
  - As we read baskets, we need to count something, e.g., occurrences of pairs.
  - The number of different things we can count is limited by the main memory.
  - Swapping counts in/out is a disaster.

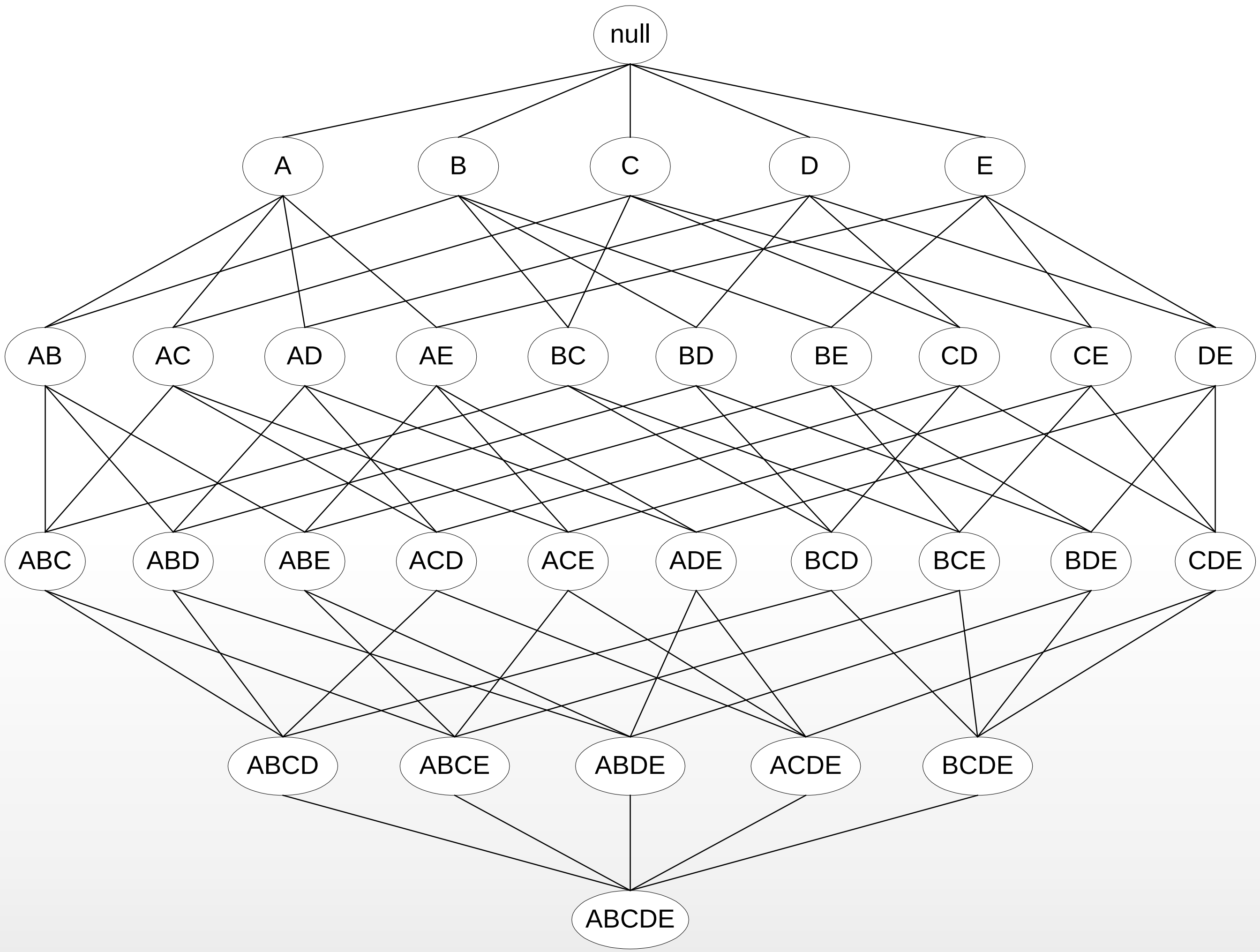
## Finding Frequent Pairs



- **The hardest problem often turns out to be finding the frequent pairs of items** $\{i_1, i_2\}$
- We'll concentrate on how to do that, then discuss extensions to finding frequent triples, etc.
- The approach:
  - We always need to generate all the itemsets
  - But we would only like to count/keep track of those itemsets that in the end turn out to be frequent

# The Lattice of Itemsets






Source: figure from Shivnath Babu, Duke CPS196.3, Lecture Notes, see
http://www.cs.duke.edu/courses/spring09/cps196.3/courseoutline.html

# Naïve Algorithm



- Naïve approach to finding frequent pairs

- Read file once, counting in main memory the occurrences of each pair:
  - From each basket of n items, generate its n(n-1)/2 pairs by two nested loops
  - Fails if $(\#items)^2$ exceeds main memory

- Note: #items can be 100K (Wal-Mart) or 10B (Web pages)
  - Suppose $10^5$ items, counts are 4-byte integers
  - Number of pairs of items: $10^5(10^5-1)/2 = 5*10^9$
  - Therefore, $2*10^{10}$ (20 gigabytes) of memory needed

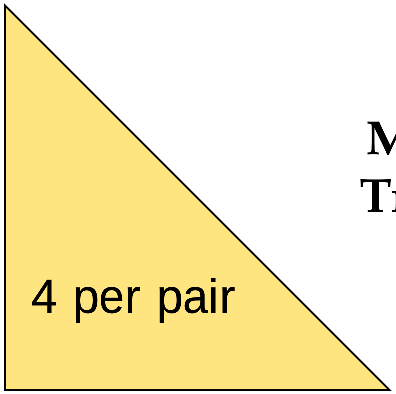
Source: Slide Adapted Jure Leskovic, Stanford CS246, Lecture Notes, see http://cs246.stanford.edu

# Details of Main-Memory Counting

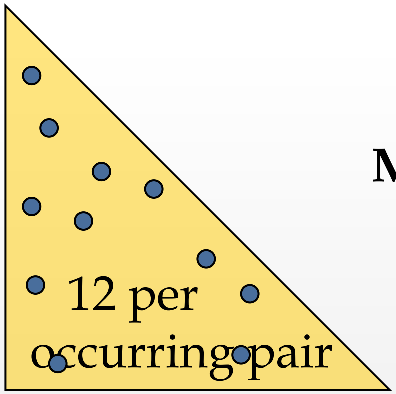


- There are two basic approaches:
  1. Count all item pairs, using a triangular matrix.
  2. Keep a table of triples [i, j, c] = the count of the pair of items {i,j } is c.

- (1) requires only (say) 4 bytes/pair; (2) requires 12 bytes, but only for those pairs with >0 counts.




**Method 1: Triangular Matrix**




**Method 2: Triples**

# The Triangular Matrix Approach



- Number items 1,2,…
- Keep pairs in the order *{1,2}, {1,3},…, {1,n }, {2,3}, {2,4},…,{2,n }, {3,4},…, {3,n },…{n -1,n }.*
- Find pair $\{i, j\}$ at the position

$$(i-1)(n-i/2)+j-i.$$

- Total number of pairs $n(n-1)/2$; total bytes about $2n^2$.

Source: Slide Adapted from Shivnath Babu, Duke CPS196.3, Lecture Notes, see earlier reference

# The Triples Approach



- You need a hash table, with $i$ and $j$ as the key, to locate $(i, j, c)$ triples efficiently.
  - Typically, the cost of the hash structure can be neglected.

- Total bytes used is about $12p$, where $p$ is the number of pairs that actually occur.
  - Beats triangular matrix if at most 1/3 of possible pairs actually occur.

# Insights/Limitations from the Previous Two Approaches



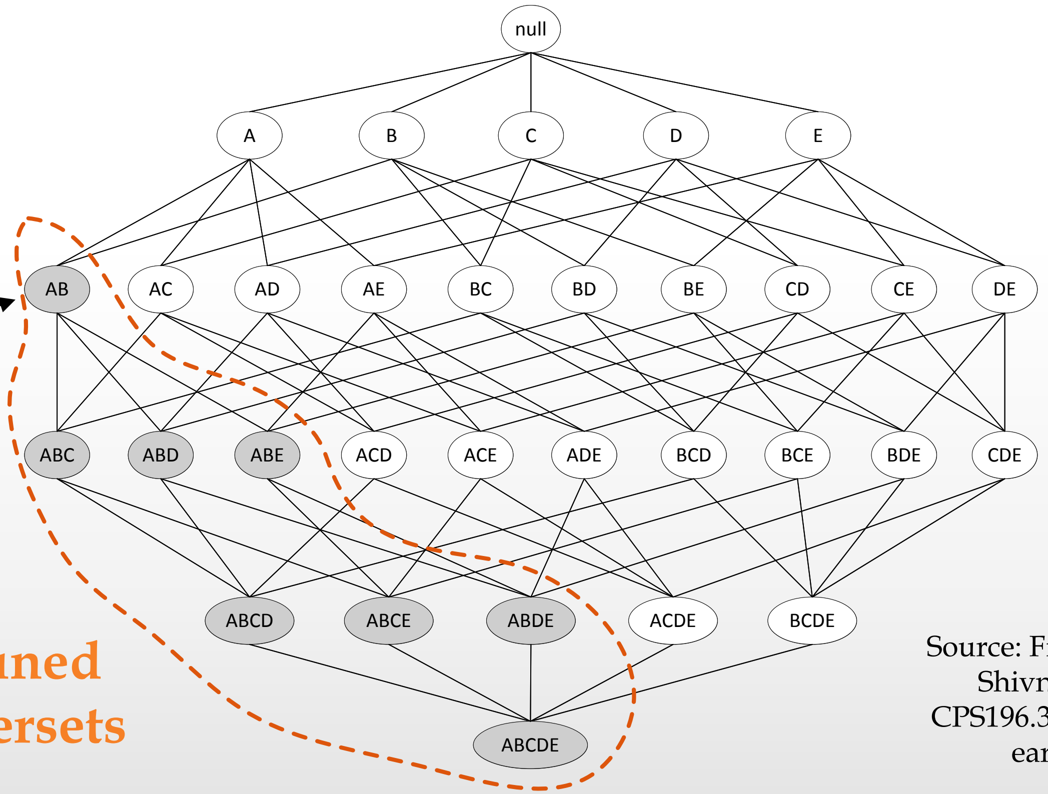
Think/Pair/Share – Activity ☺

# The A-Priori Algorithm



- Key idea: ***monotonicity*** - if a set of items appears at least $s$ times, so does every subset.
  - **Contrapositive for pairs:** if item $i$ does not appear in $s$ baskets, then no pair including $i$ can appear in $s$ baskets.

**Found to be Infrequent**




**Pruned supersets**

Source: Figure Adapted from Shivnath Babu, Duke CPS196.3, Lecture Notes, see earlier reference

# Illustrating the A-priori Principle*



- Consider the following market-basket data

| *TID* | *Items* |
|---|---|
| **1** | **Bread, Milk** |
| **2** | **Bread, Diaper, Beer, Eggs** |
| **3** | **Milk, Diaper, Beer, Coke** |
| **4** | **Bread, Milk, Diaper, Beer** |
| **5** | **Bread, Milk, Diaper, Coke** |

* Example from Shivnath Babu, Duke CPS196.3, Lecture Notes, see earlier reference

# Illustrating the A-priori Principle



| Item | Count |
|---|---|
| **Bread** | **4** |
| **Coke** | **2** |
| **Milk** | **4** |
| **Beer** | **3** |
| **Diaper** | **4** |
| **Eggs** | **1** |

Items (1-itemsets)

| Itemset | Count |
|---|---|
| **{Bread,Milk}** | **3** |
| **{Bread,Beer}** | **2** |
| **{Bread,Diaper}** | **3** |
| **{Milk,Beer}** | **2** |
| **{Milk,Diaper}** | **3** |
| **{Beer,Diaper}** | **3** |

Pairs (2-itemsets)

(No need to generate candidates involving Coke or Eggs)

Triplets (3-itemsets)

| Itemset | Count |
|---|---|
| **{Bread,Milk,Diaper}** | **3** |

Minimum Support = 3

If every subset is considered,

$^6C_1 + {}^6C_2 + {}^6C_3 = 41$

With support-based pruning,

$6 + 6 + 1 = 13$

# Details of the A-Priori Algorithm



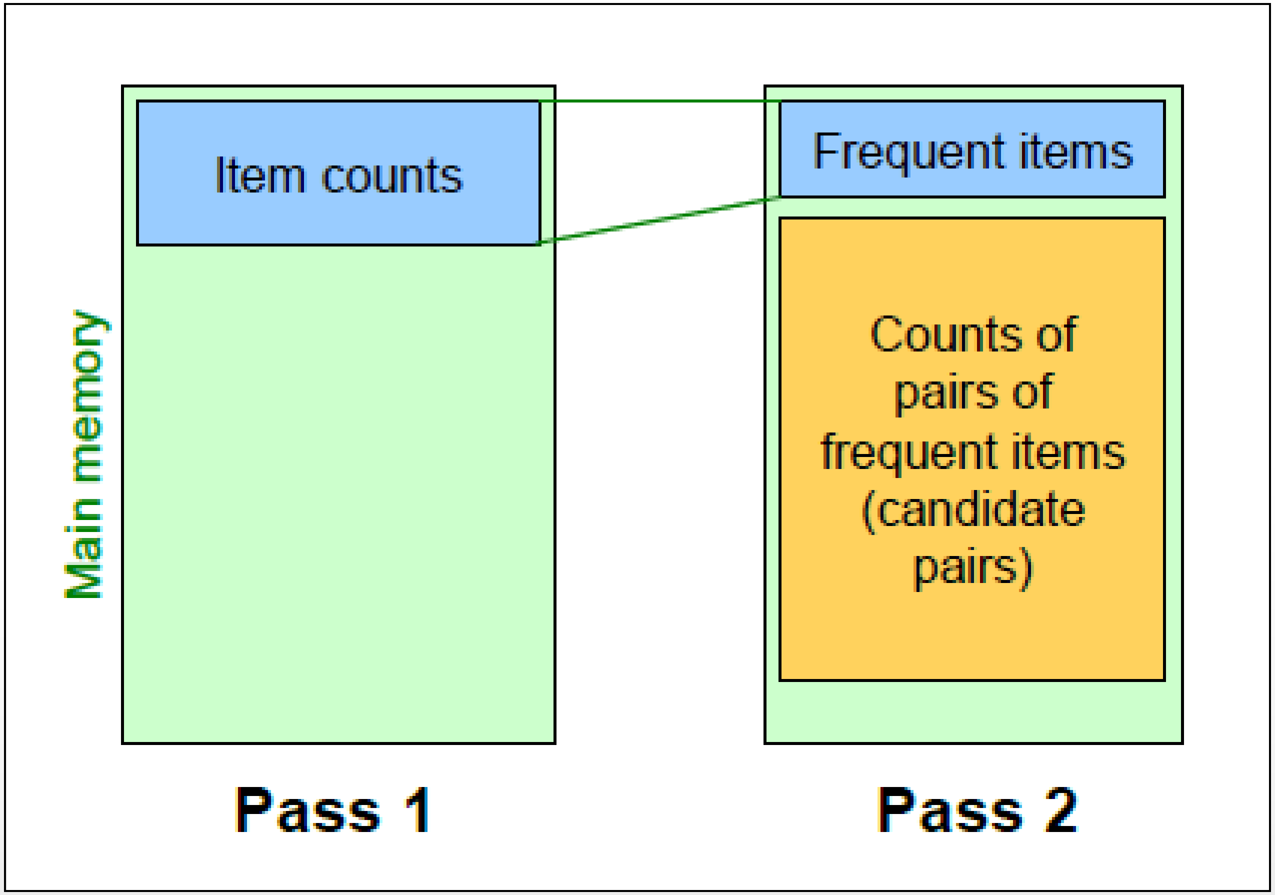
- Pass 1: Read baskets and count in main memory the occurrences of each item.
  - Requires only memory proportional to #items.

- Pass 2: Read baskets again and count in main memory only those pairs both of which were found in Pass 1 to be frequent.
  - Requires memory proportional to square of frequent items only.
  - Plus a list of the frequent items (so you know what must be counted)

Source: Slide Adapted Jure Leskovic, Stanford CS246, Lecture Notes, see http://cs246.stanford.edu

# Main Memory: Picture of the A-Priori

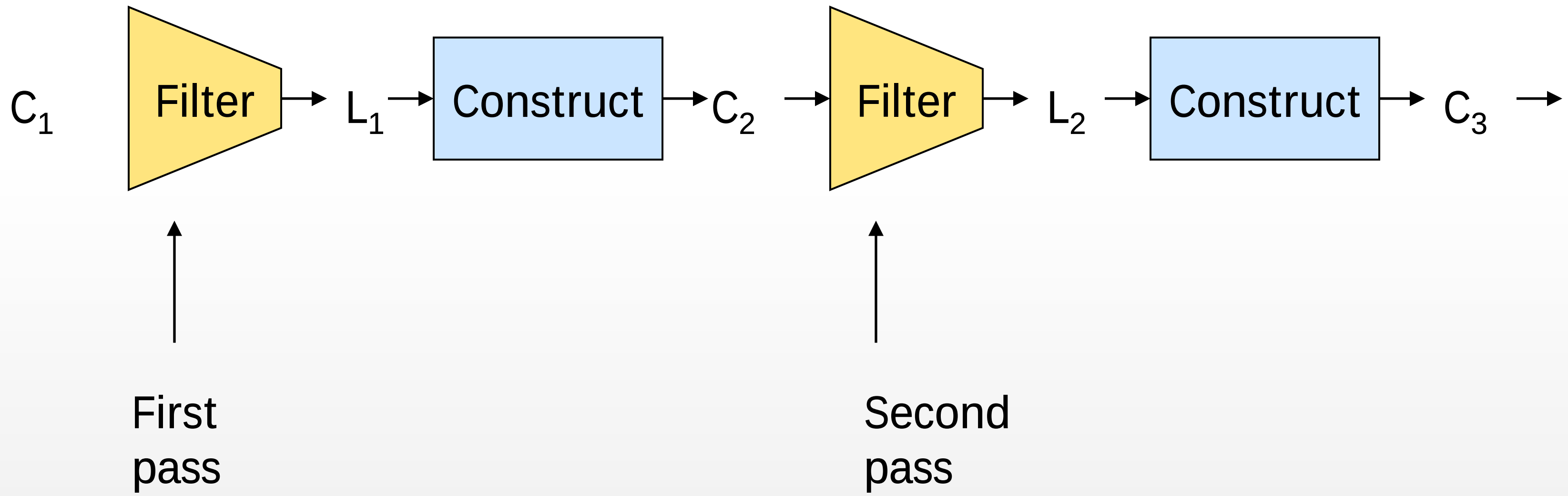





Source: Slide Adapted Jure Leskovic, Stanford CS246, Lecture Notes, see http://cs246.stanford.edu

# Frequent Triples, Etc.

- For each $k$, we construct two sets of $k$ –tuples:
  - $C_k$ = **candidate** $k$ – tuples = those that might be frequent sets (support $\geq s$ ) based on information from the pass for $k$ –1.
  - $L_k$ = the set of truly frequent $k$ –tuples.




Source: Slide Adapted Jure Leskovic, Stanford CS246, Lecture Notes, see http://cs246.stanford.edu

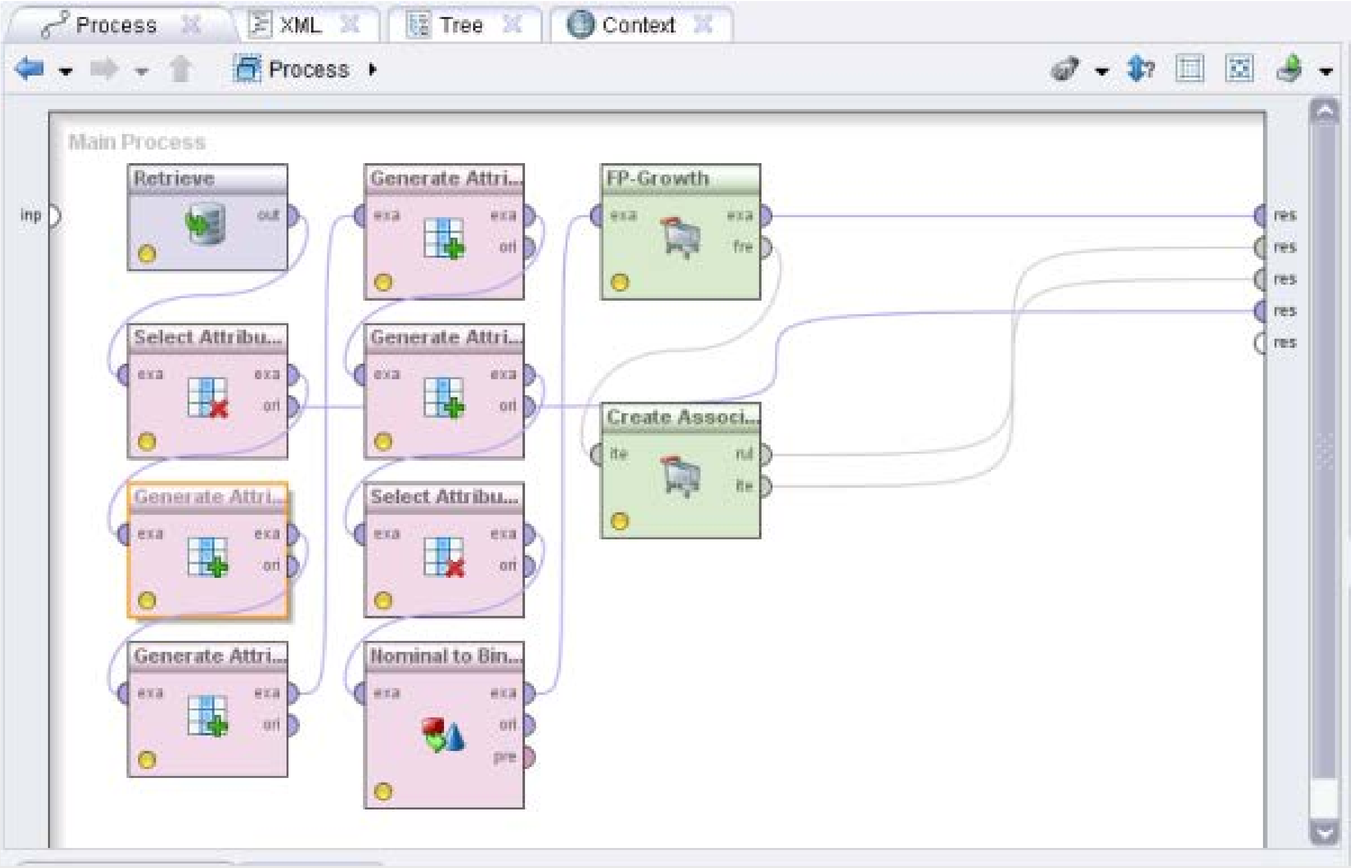
# Example

- **Hypothetical steps of the A-Priori algorithm**
- $C_1$ = { {b} {c} {j} {m} {n} {p} }
- Count the support of itemsets in $C_1$
- Prune non-frequent: $L_1$ = { b, c, j, m }
- Generate $C_2$ = { {b,c} {b,j} {b,m} {c,j} {c,m} {j,m} }
- Count the support of itemsets in $_{C2}$
- Prune non-frequent: $L_2$ = { {b,m} {b,c} {c,m} {c,j} }
- Generate $C_3$ = { {b,c,m} {b,c,j} {b,m,j} {c,m,j} }
- Count the support of itemsets in $C_3$
- Prune non-frequent: $L_3$ = { {b,c,m} }

Source: Slide Adapted Jure Leskovic, Stanford CS246, Lecture Notes, see http://cs246.stanford.edu

# Using Software for Generating Association Rules - Patrick

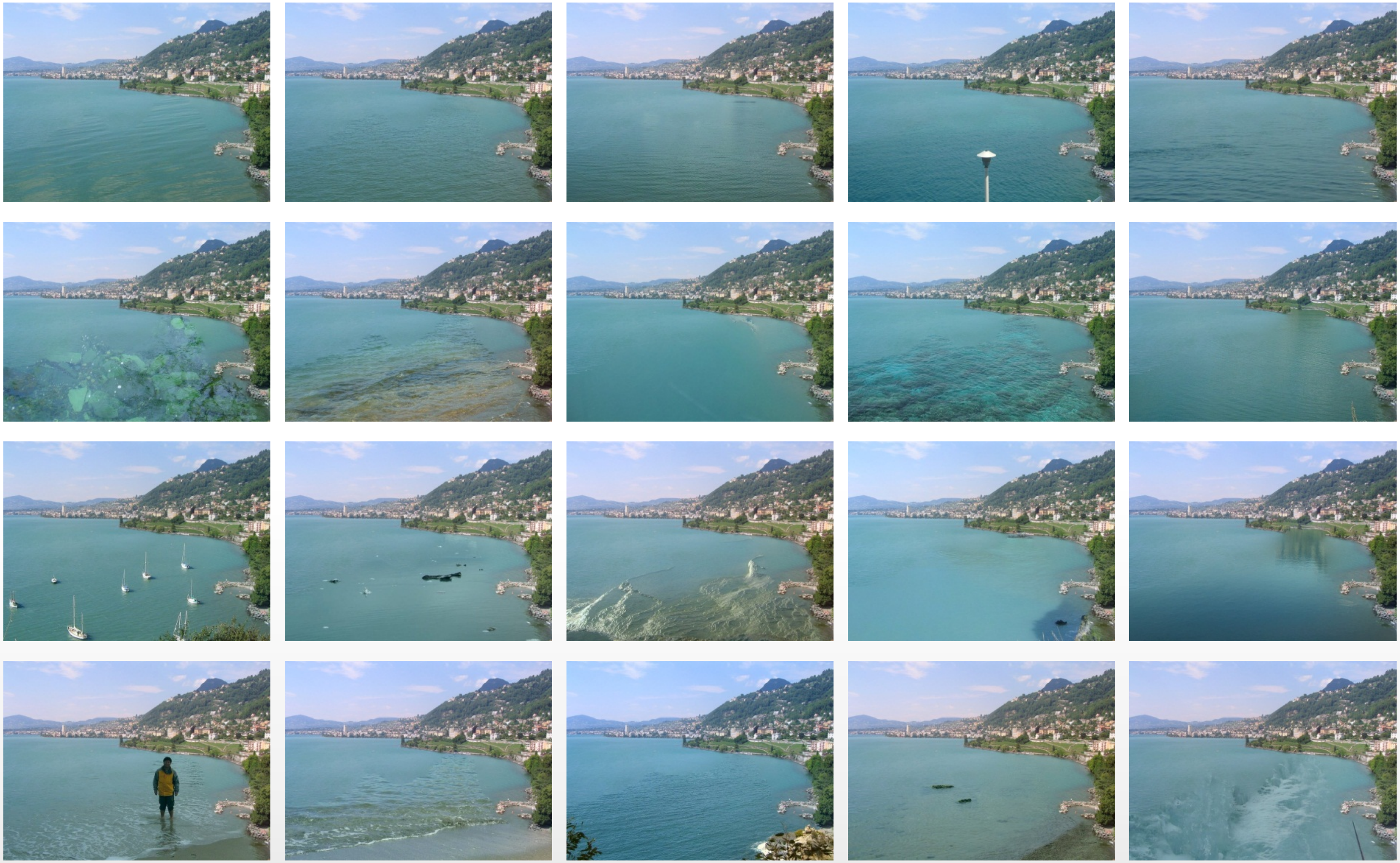
- A demonstration on how to generate association rules using software ☺



# Chapter 03:
# Finding Similar Items

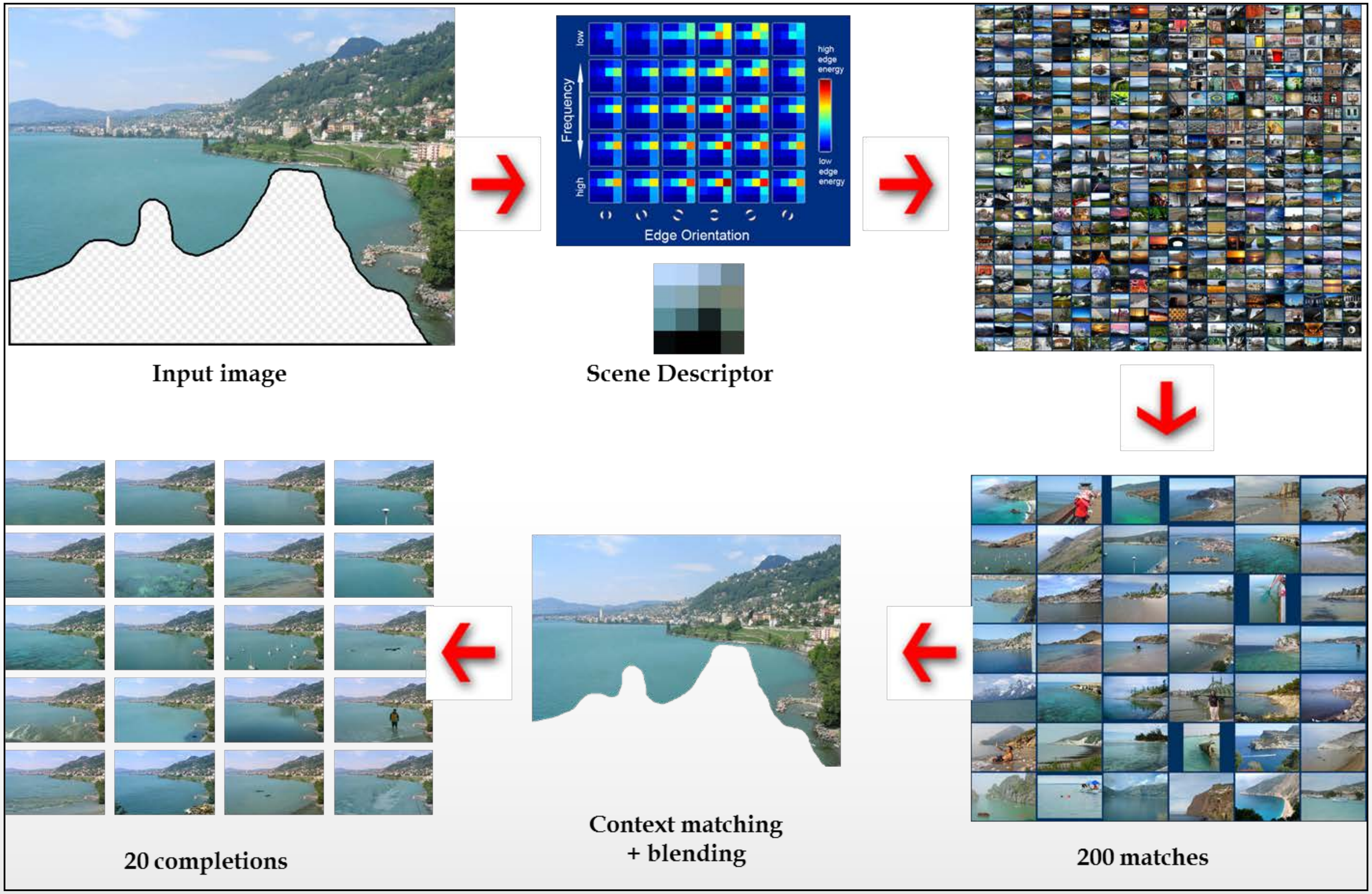
# Some Introductory Remarks – What is “similar”?





Source: Figure from James Hays, Alexei A. Efros. Scene Completion Using Millions of Photographs. ACM Transactions on Graphics (SIGGRAPH 2007). August 2007, vol. 26, No. 3. http://graphics.cs.cmu.edu/projects/scene-completion/

# One Application of Similarity

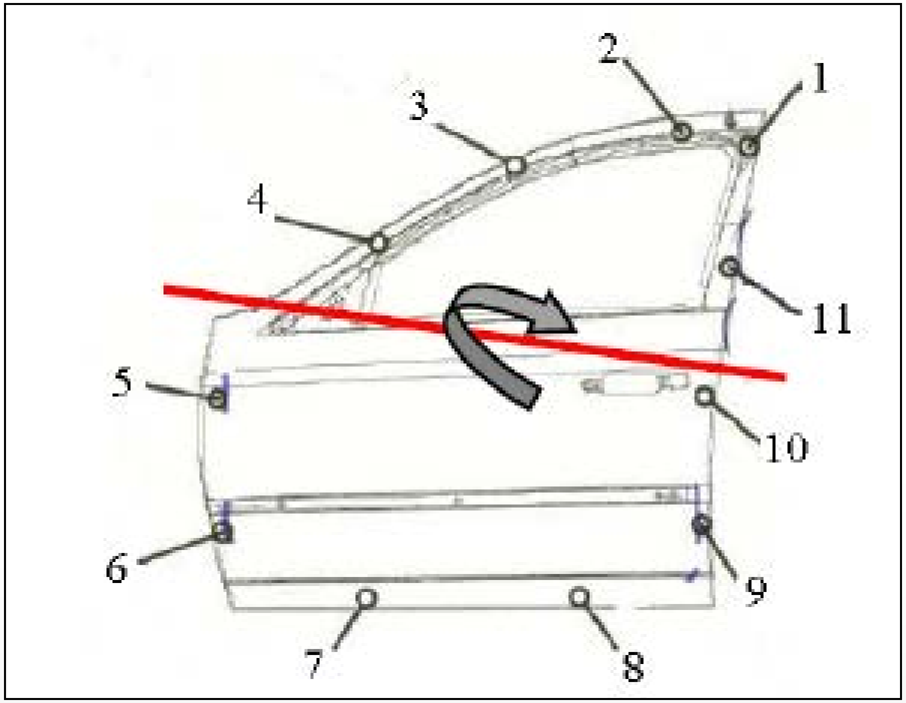



Source: Figure from James Hays, Alexei A. Efros. Scene Completion Using Millions of Photographs. ACM Transactions on Graphics (SIGGRAPH 2007). August 2007, vol. 26, No. 3. http://graphics.cs.cmu.edu/projects/scene-completion/

# An Industrial Engineering Application of Similarity



- Web Mining: Detect duplicate pages and mirror sites to improve search efficiency.
- Manufacturing: Detect data redundancy to increase the efficiency of the quality assessment
  - Example: Door gap




Decision on the number of sensors for detecting a frequent defect in manufacturing operations (true example)

# So How is Similarity and Association Different?



- They are definitely related concepts. However, there is a relatively small difference between them ☺

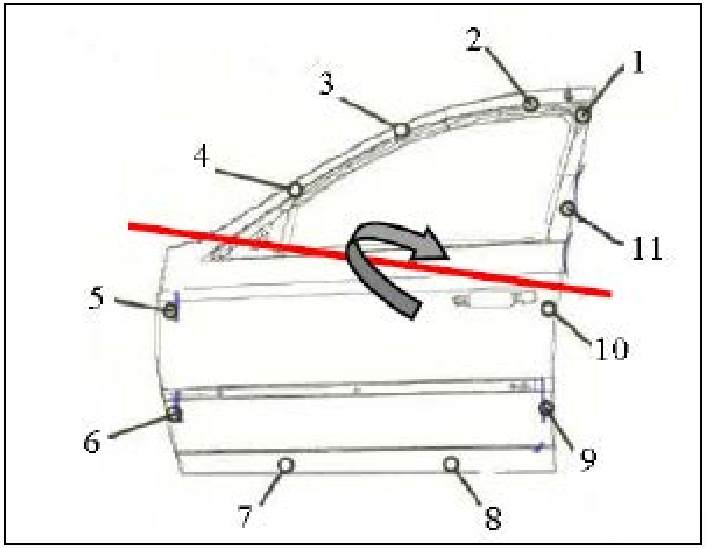
- **Book Definition**: "the problem of finding frequent itemsets differs from the similarity search…"
  - In frequent itemsets, we are interested in the absolute # of baskets that contain a particular set of items
  - For similarity, we want items that have a large fraction of their basket in common, even if the absolute # of baskets is small.

- IE take on the difference?? – see next slide ☺

# So How is Similarity and Association Different?

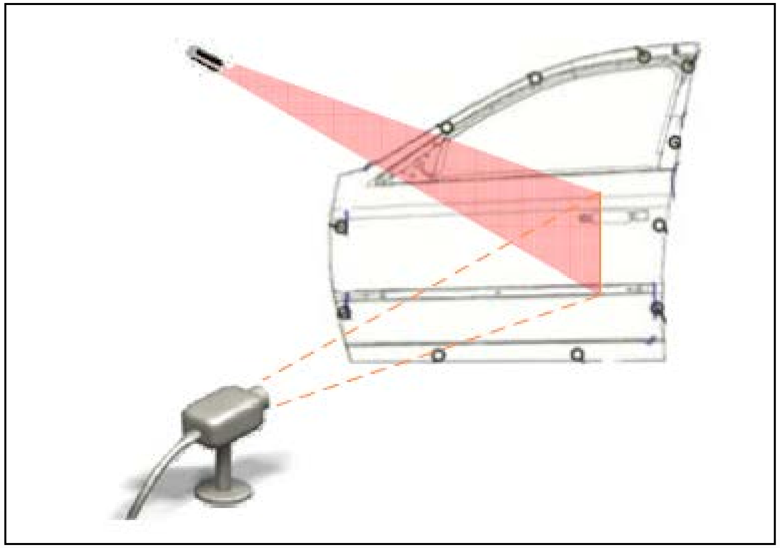


## Similarity




**Sensors 1&2** providing redundant information to detect a certain fault

## Association Rules



If a strong association is present between the 3D scanner and the CMM, it may be possible to infer the optimal placement for gap measurements from inherent associations within the data.

# Analytics and Visualization of Big Data

*Fadel M. Megahed*

## *Lecture 09: Association Rules and Similarity*

AUBURN UNIVERSITY
SAMUEL GINN
COLLEGE OF ENGINEERING

**Department of Industrial and Systems Engineering**

*Spring 13*