MIE524 Data Mining

Mining Association Rules

Slides Credits:
Slides from Leskovec, Rajaraman, Ullman (http://www.mmds.org),

MIE524: Course Topics (Tentative)

Large-scale Machine Learning	High-dimensional Data	Graph Data	Applications
Learning Embedding (NN / AE)	Locality sensitive hashing	Processing Massive Graphs	Recommender systems
Decision Trees	Clustering	PageRank, SimRank	Association Rules
Ensemble Models (GBTs)	Dimensionality reduction	Graph Representation Learning	Neural Language Models

Computational Models: Single Machine | MapReduce/Spark | GPU

Association Rule Discovery

Supermarket shelf management – Market-basket model:

- **Goal:** Identify items that are bought together by sufficiently many customers
- **Approach:** Process the sales data collected with barcode scanners to find dependencies among items
- **A classic rule:**
  - If someone buys diaper and milk, then he/she is likely to buy beer
  - Don't be surprised if you find six-packs next to diapers!

The Market-Basket Model

- A **large set** of **items**
  - e.g., things sold in a supermarket
- A **large set** of **baskets**
  - Each basket is a **small subset of items**
    - e.g., the things one customer buys on one day
- **Discover association rules:**
  People who bought {x,y,z} tend to buy {v,w}
    - Example application: Amazon

Input:

Basket	*Items*
1	Bread, Coke, Milk
2	Beer, Bread
3	Beer, Coke, Diaper, Milk
4	Beer, Bread, Diaper, Milk
5	Coke, Diaper, Milk

Output:

Rules Discovered:
{Milk} --> {Coke}
{Diaper, Milk} --> {Beer}

More generally

- **A general many-to-many mapping (association) between two kinds of things**
  - But we ask about connections among “items”, not “baskets”
- **Items and baskets are abstract:**
  - **For example:**
    - Items/baskets can be products/shopping basket
    - Items/baskets can be words/documents
    - Items/baskets can be base-pairs/genes
    - Items/baskets can be drugs/patients

Applications – (1)

- **Items** = products; **Baskets** = sets of products someone bought in one trip to the store
- **Real market baskets:** Chain stores keep TBs of data about what customers buy together
  - Tells how typical customers navigate stores, lets them position tempting items together:
    - Apocryphal story of "diapers and beer" discovery
    - Used to position potato chips between diapers and beer to enhance sales of potato chips
- ***Amazon's 'people who bought X also bought Y'***

Applications – (2)

- **Baskets** = sentences; **Items** = documents in which those sentences appear
  - Items that appear together too often could represent plagiarism
  - Notice items do not have to be "in" baskets
- **Baskets** = patients; **Items** = drugs & side-effects
  - Has been used to detect combinations of drugs that result in particular side-effects
  - **But requires extension:** Absence of an item needs to be observed as well as presence

Outline

First: Define

- **Frequent itemsets**
- **Association rules:**
  - Confidence, Support, Interestingness

Then: Algorithms for finding frequent itemsets

- **Finding frequent pairs**
- **A-Priori algorithm**
- **PCY algorithm**

Frequent Itemsets

- **Simplest question:** Find sets of items that appear together "frequently" in baskets
- ***Support*** for itemset I: Number of baskets containing all items in I
  - (Often expressed as a fraction of the total number of baskets)
- Given a ***support threshold*** s, then sets of items that appear in at least s baskets are called ***frequent itemsets***

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

Support of {Beer, Bread} = 2

Example: Frequent Itemsets

- **Items** = {milk, coke, pepsi, beer, juice}
- **Support threshold** = 3 baskets

B_1 = {m, c, b} B_2 = {m, p, j}

B_3 = {m, b} B_4 = {c, j}

B_5 = {m, p, b} B_6 = {m, c, b, j}

B_7 = {c, b, j} B_8 = {b, c}

- **Frequent itemsets:** {m}, {c}, {b}, {j}, {m,b} , {b,c} , {c,j}.

Define: Association Rules

- **Define: Association Rules:**
  If-then rules about the contents of baskets
- $\{i_1, i_2,\ldots,i_k\} \rightarrow j$ means: "if a basket contains all of $i_1,\ldots,i_k$ then it is ***likely*** to contain j"
- **In practice there are many rules, want to find significant/interesting ones!**
- ***Confidence*** of association rule is the probability of j given $I = \{i_1,\ldots,i_k\}$

$$\text{conf}(I \rightarrow j) = \frac{\text{support}(I \cup j)}{\text{support}(I)}$$

$$\text{conf}(I \rightarrow j) = \\ = P(j|I) = \frac{P(I,j)}{P(I)}$$

Interesting Association Rules

- **Not all high-confidence rules are interesting**
  - The rule $X \rightarrow \textit{milk}$ may have high confidence for many itemsets X, because milk is just purchased very often (independent of X)
- **Interest** of an association rule $I \rightarrow j$: abs. difference between its confidence and the fraction of baskets that contain j

$$Interest(\mathrm{I} \rightarrow j) = |conf(I \rightarrow j) - \mathrm{P}[j]| = |P(j|I) - P(j)|$$

  - Interesting rules: those with high interest values (usually above 0.5)
  - Why absolute value? Want to capture both *positive* and *negative* associations between itemsets and items

Example: Confidence and Interest

$B_1 = \{m, c, b\}$ $B_2 = \{m, p, j\}$
$B_3 = \{m, b\}$ $B_4 = \{c, j\}$
$B_5 = \{m, p, b\}$ $B_6 = \{m, c, b, j\}$
$B_7 = \{c, b, j\}$ $B_8 = \{b, c\}$

- **Association rule: $\{m, b\} \rightarrow c$**
  - **Support** = 2
  - **Confidence** = 2/4 = 0.5
  - **Interest** = |0.5 – 5/8| = 1/8
    - Item ***c*** appears in 5/8 of the baskets
    - The rule is not very interesting!

Association Rule Mining

- **Problem: Find all association rules with support $\geq s$ and confidence $\geq c$**
  - **Note:** Support of an association rule is the support of the set of items in the rule (left and right side)
- **Hard part: Finding the frequent itemsets!**
  - If $\{i_1, i_2, \ldots, i_k\} \rightarrow j$ has high support and confidence, then both $\{i_1, i_2, \ldots, i_k\}$ and $\{i_1, i_2, \ldots, i_k, j\}$ will be "frequent"

$$\text{conf}(I \rightarrow j) = \frac{\text{support}(I \cup j)}{\text{support}(I)}$$

Mining Association Rules

$$\text{conf}(I \to j) = \frac{\text{support}(I \cup j)}{\text{support}(I)}$$

- **Step 1:** Find all frequent itemsets I
  - (we will explain this next)
- **Step 2: Rule generation**
  - For every subset A of I, generate a rule $A \to I \setminus A$
    - Since I is frequent, A is also frequent
    - **Variant 1:** Single pass to compute the rule confidence
      - confidence(***A,B***$\rightarrow$***C,D***) = support(**A,B,C,D**) / support(**A,B**)
    - **Variant 2:**
      - **Observation:** If **A,B,C**$\rightarrow$**D** is below confidence, then so is **A,B**$\rightarrow$**C,D**
      - Can generate "bigger" rules from smaller ones!
  - **Output the rules above the confidence threshold**

Example

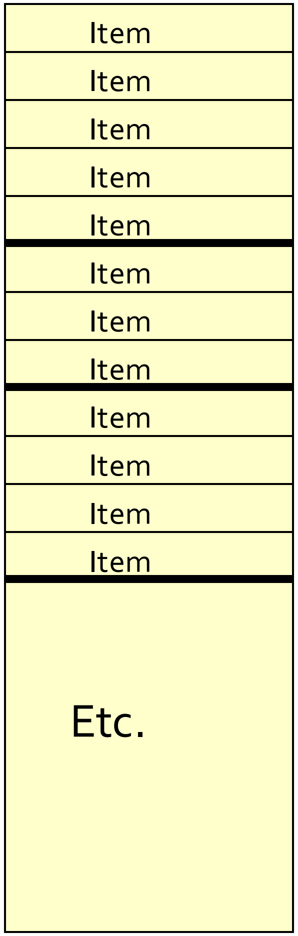
$B_1 = \{m, c, b\}$ $B_2 = \{m, p, j\}$
$B_3 = \{m, c, b, n\}$ $B_4 = \{c, j\}$
$B_5 = \{m, p, b\}$ $B_6 = \{m, c, b, j\}$
$B_7 = \{c, b, j\}$ $B_8 = \{b, c\}$

- Support threshold $s = 3$, confidence $c = 0.75$
- Step 1) Find frequent itemsets:
  - {b,m} {b,c} {c,m} {c,j} {m,c,b}
- Step 2) Generate rules:
  - ~~b→m: c=4/6~~ b→c: c=5/6 ~~b,c→m: c=3/5~~
  - m→b: c=4/5 … b,m→c: c=3/4
    ~~b→c,m: c=3/6~~

Step 2:
Finding Frequent Itemsets

Itemsets: Computation Model

- **Back to finding frequent itemsets**
- Typically, data is kept in flat files rather than in a database system:
  - Stored on disk
  - Stored basket-by-basket
  - Baskets are **small** but we have many baskets and many items
    - Expand baskets into pairs, triples, etc. as you read baskets
    - Use ***k*** nested loops to generate all sets of size ***k***

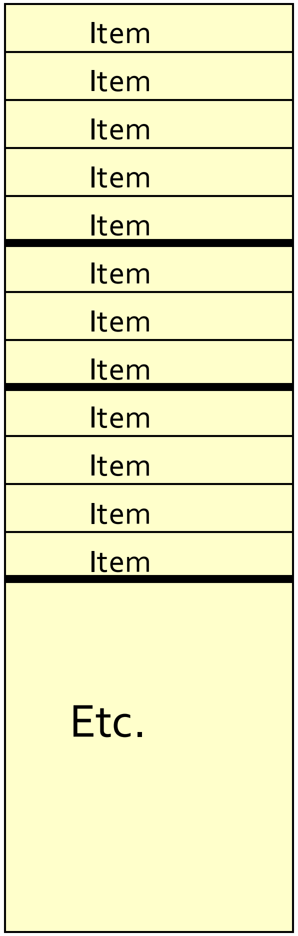


Items are positive integers, and boundaries between baskets are –1.

Note: We want to find frequent itemsets. To find them, we have to count them. To count them, we have to enumerate them.

Computation Model

- The true cost of mining disk-resident data is usually the **number of disk I/Os**
- In practice, association-rule algorithms read the data in ***passes***
  – all baskets read in turn
- We measure the cost by the **number of passes** an algorithm makes over the data



Items are positive integers, and boundaries between baskets are –1.

Main-Memory Bottleneck

- For many frequent-itemset algorithms, **main-memory** is the critical resource
  - As we read baskets, we need to count something, e.g., occurrences of pairs of items
  - The number of different things we can count is limited by 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 $\{i_1, i_2\}$**
  - **Why?** Freq. pairs are common, freq. triples are rare
    - **Why?** Probability of being frequent drops exponentially with size; number of sets grows more slowly with size

- **Let's first concentrate on pairs, then extend to larger sets**

Finding Frequent Pairs

- **The approach:**
  - We always need to "generate" all the itemsets
  - But we would only like to count (keep track of) only those itemsets that in the end turn out to be frequent
- **Scenario:**
  - Imagine we aim to identify frequent pairs
  - We will need to enumerate all pairs of items
    - For every basket, enumerate all pairs of items in that basket
  - **But,** rather than keeping a count for every pair, we hope to discard a lot of pairs and only keep track of the ones that will in the end turn out to be frequent

Naïve Algorithm

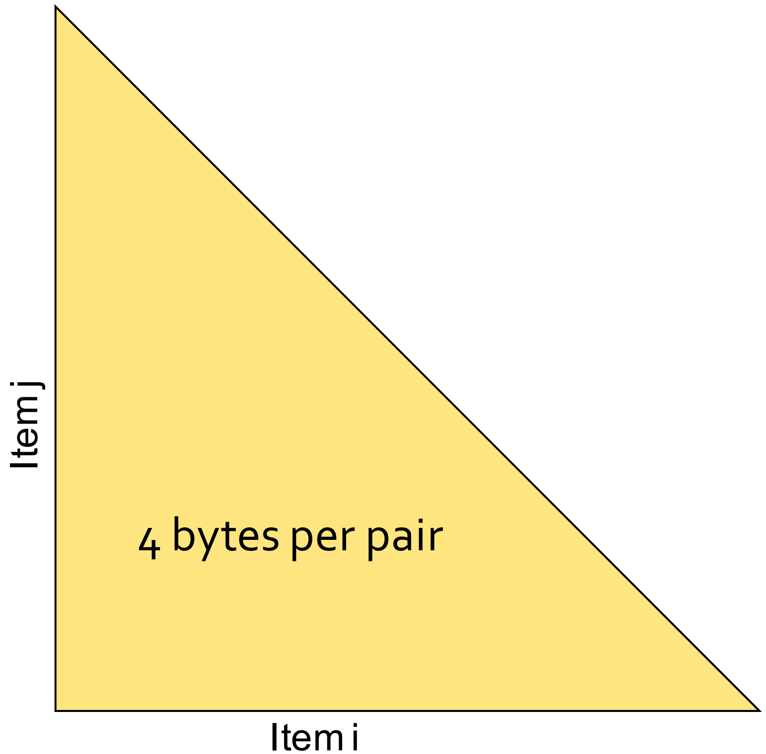
- **Naïve approach to finding frequent pairs**
- Read file once, counting in main memory the occurrences of each pair:
  - From each basket ***b*** of n_b items, generate its $n_b(n_b\text{-}1)/2$ pairs by two nested loops
  - A data structure then keeps count of every pair
- **Fails if (#items)^2 exceeds main memory**
  - **Remember:** #items can be 100K (Wal-Mart) or 10B (Web pages)
    - Suppose 10^5 items, counts are 4-byte integers
    - Number of pairs of items: $10^5(10^5-1)/2 \approx 5*10^9$
    - Therefore, $2*10^{10}$ (20 gigabytes) of memory is needed

Counting Pairs in Memory

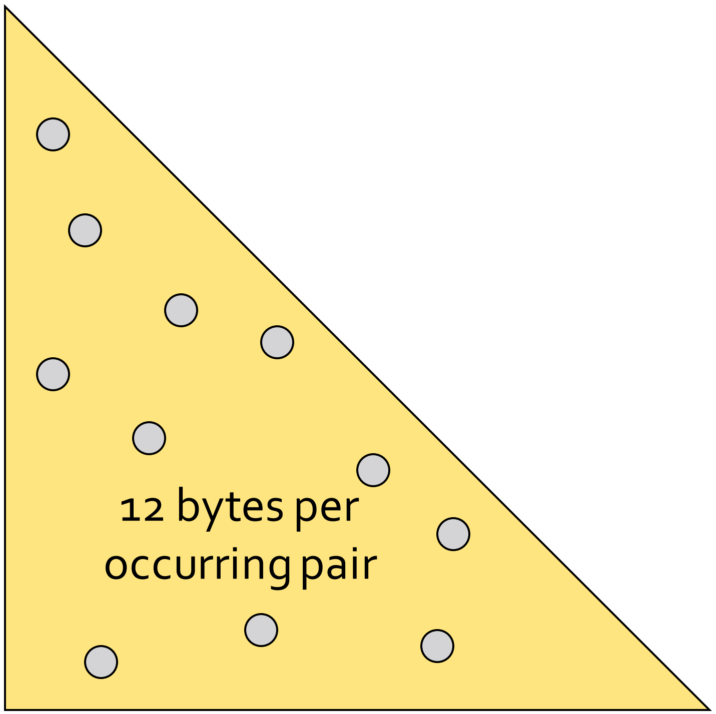
Goal: Count the number of occurrences of each pair of items *(i,j)*:

- **Approach 1:** Count all pairs using a matrix
- **Approach 2:** Keep a table of triples [*i*, *j*, *c*] = "the count of the pair of items {*i*, *j*} is *c*."
  - If integers and item ids are 4 bytes, we need approximately 12 bytes for pairs with count > 0
  - Plus some additional overhead for the hashtable

Comparing the Two Approaches



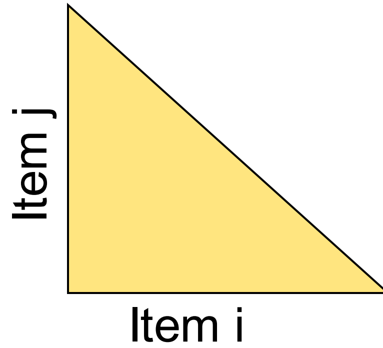
Triangular Matrix



Triples (item i, item j, count)

Comparing the Two Approaches

- **Approach 1: Triangular Matrix**
  - **n** = total number items
  - Count pair of items $\{i, j\}$ only if $i<j$
  - Keep pair counts in lexicographic order:
    - {1,2}, {1,3},…, {1,*n*}, {2,3}, {2,4},…,{2,*n*}, {3,4},…
  - Pair $\{i, j\}$ is at position: $[n(n-1) - (n-i)(n-i+1)]/2 + (j-i)$
  - Total number of pairs $n(n-1)/2$; total bytes= $O(n^2)$
  - **Triangular Matrix** requires 4 bytes per pair

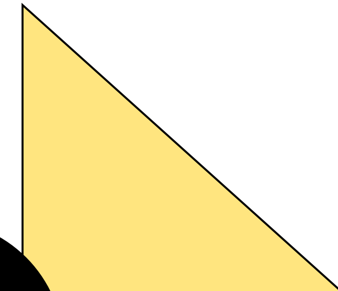


- **Approach 2** uses **12 *bytes*** per occurring pair *(but only for pairs with count > 0)*

- Approach 2 beats Approach 1 if less than **1/3** of possible pairs actually occur

Comparing the Two Approaches

- **Approach 1: Triangular Matrix**
  - **n** = total number items
  - Co
  - K
  - F **2 + (j - i)**
  - T)
  - T



Problem is when we have too many items so all the pairs do not fit into memory.

Can we do better?

- **Ap**
  (bu

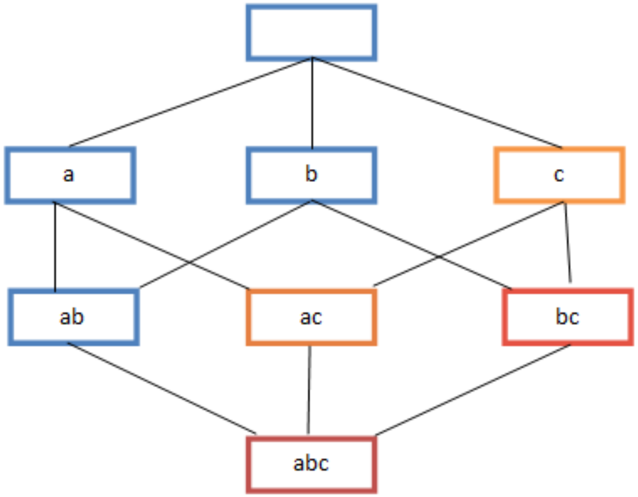
- Approach 2 beats Approach 1 if less than **1/3** of possible pairs actually occur

A-Priori Algorithm

Key concepts:

- **Monotonicity of “Frequent”**
- **Notion of Candidate Pairs**
- **Extension to Larger Itemsets**

A-Priori Algorithm – (1)

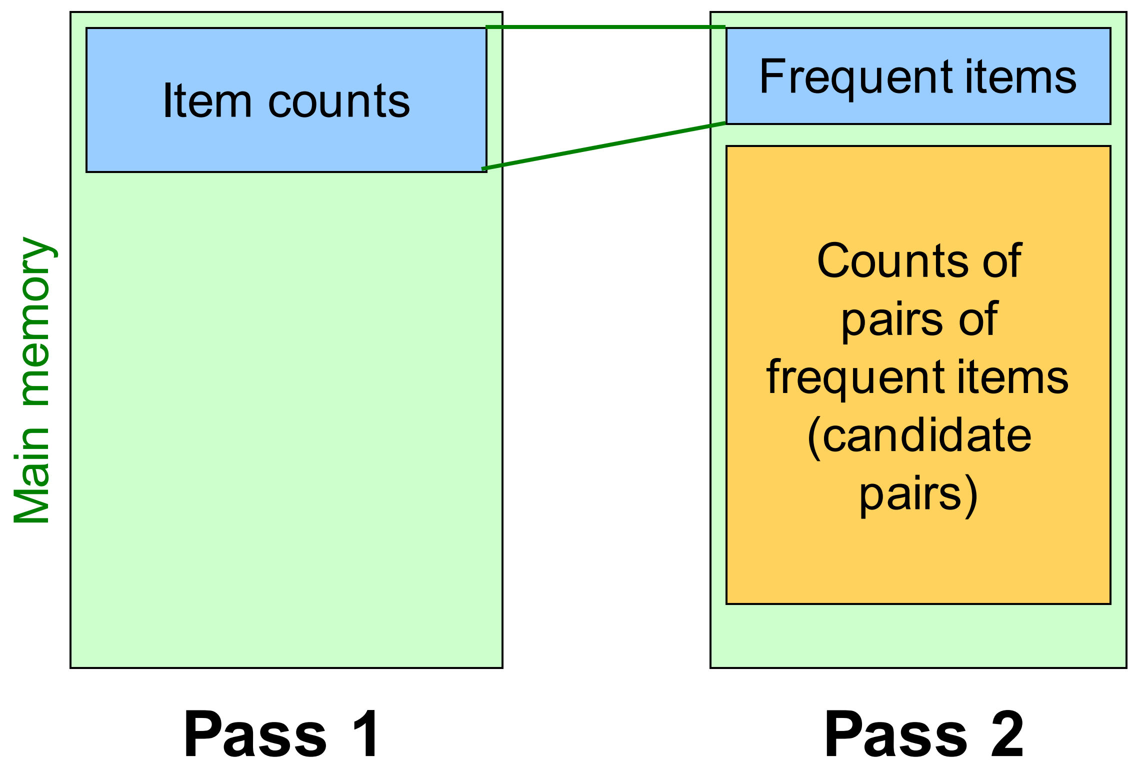


- A **two-pass** approach called ***A-Priori*** limits the need for main memory
- **Key idea: *monotonicity***
  - If a set of items I appears at least s times, so does every **subset** J of I
- **Contrapositive for pairs:**
  If item i does not appear in s baskets, then no pair including i can appear in s baskets
- **So, how does A-Priori find freq. pairs?**

A-Priori Algorithm – (2)

- **Pass 1:** Read baskets and count in main memory the # of occurrences of each **individual item**
  - Requires only memory proportional to #items
- **Items that appear $\geq s$ times are the frequent items**
- **Pass 2:** Read baskets again and keep track of the count of only those pairs where both elements are frequent (from Pass 1)
  - Requires memory (for counts) proportional to square of the number of **frequent** items (not the square of total # of items)
  - Plus a list of the frequent items (so you know what must be counted)

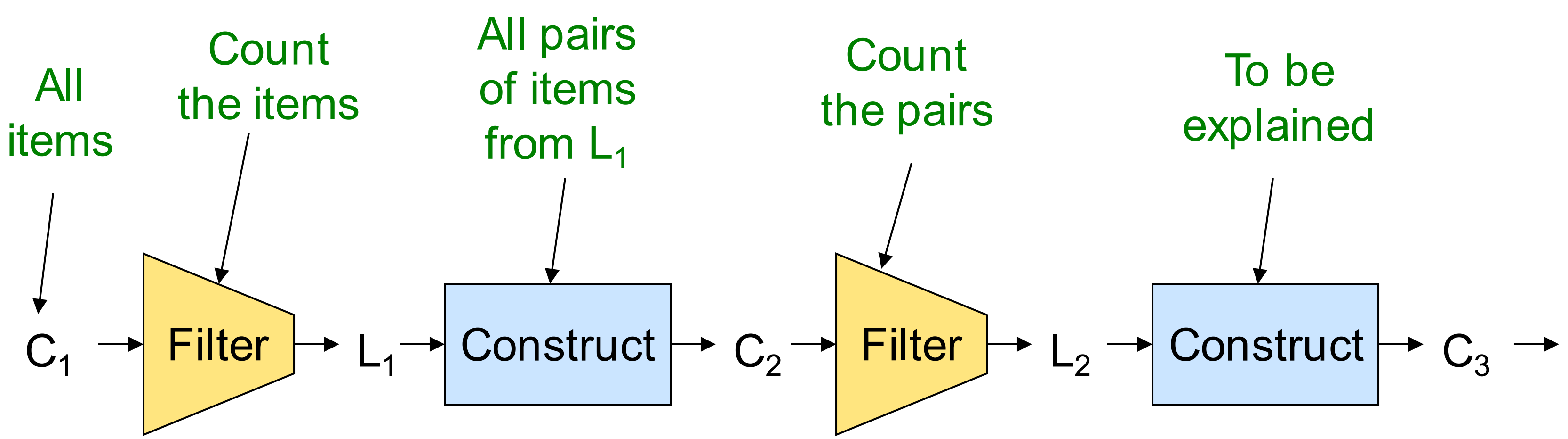
Main-Memory: Picture of A-Priori



Green box represents the amount of available main memory. Smaller boxes represent how the memory is used.

Frequent Triples, Etc.

- **For each *k*, we construct two sets of *k-tuples*** (sets of size *k*):
  - ***C***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



Example

** Note here we generate new candidates by generating C_k from L_{k-1} and L_1.
But one can be more careful with candidate generation. For example, in C_3 we know {b,m,j} cannot be frequent since {m,j} is not frequent

- **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. We get: 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 C_2
  - Prune non-frequent. L_2 = { {b,m} {b,c} {c,m} {c,j} }
  - Generate C_3 = { {b,c,m} <u>{b,c,j} {b,m,j} {c,m,j}</u> } **
  - Count the support of itemsets in C_3
  - Prune non-frequent. L_3 = { {b,c,m} }

A-Priori for All Frequent Itemsets

- One pass for each ***k*** (itemset size)
- Needs room in main memory to count each candidate ***k***–tuple
- For typical market-basket data and reasonable support (e.g., 1%), ***k* = 2** requires the most memory