Frequent Itemset Mining & Association Rules

Mining of Massive Datasets

Jure Leskovec, Anand Rajaraman, Jeff Ullman
Stanford University

http://www.mmds.org

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
- Want to discover **association rules**
  - People who bought {x,y,z} tend to buy {v,w}
    - Amazon!

Input:

TID	*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}

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
  - Suggests tie-in “tricks”, e.g., run sale on diapers and raise the price of beer
  - Need the rule to occur frequently, or no $$’s
- **Amazon’s people who bought *X* also bought *Y***

Applications – (2)

- **Baskets** = sentences; **Items** = documents containing those sentences
  - 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

Applications – (3)

- **Baskets** = Documents; **Items** = words
  - Unusual words appearing in a large number of documents, e.g. “Brad” and “Angelina” may indicate an interesting relationship.

More generally

- **A general many-to-many mapping (association) between two kinds of things**
  - But we ask about connections among “items”, not “baskets”

Scale of the Problem

- WalMart sells 100k items and can store billions of basket
- Web has billions of words and many billions of pages

Frequent Itemsets

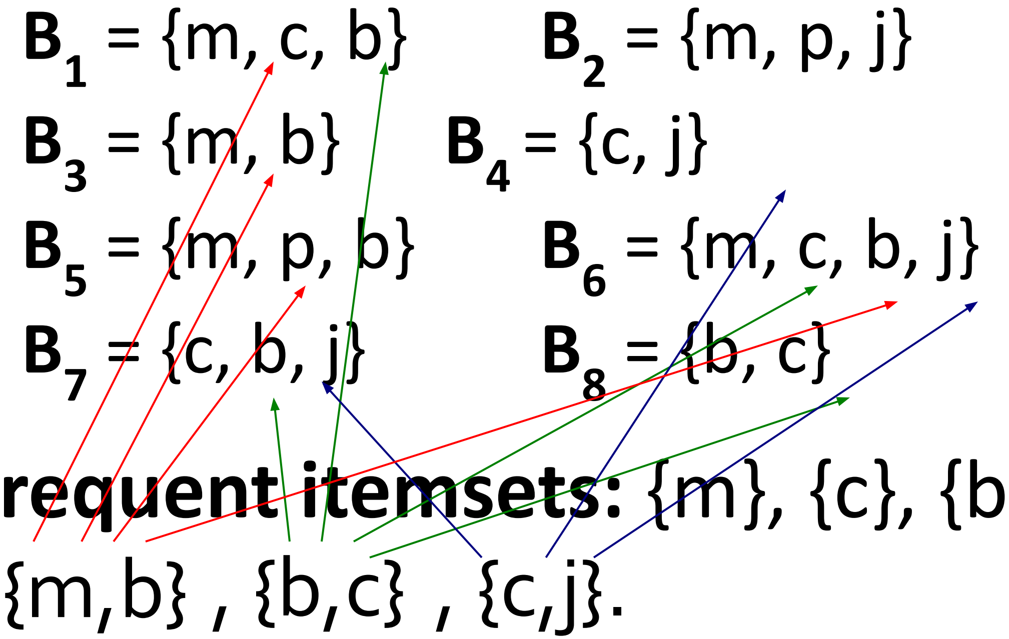
- **Simplest question:** Find sets of items that appear together “frequently” in baskets
- ***Support*** for itemset I: Number of baskets containing all items in I
  - (Often expressed as a fraction of the total number of baskets)
- Given a ***support threshold*** s, then sets of items that appear in 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}

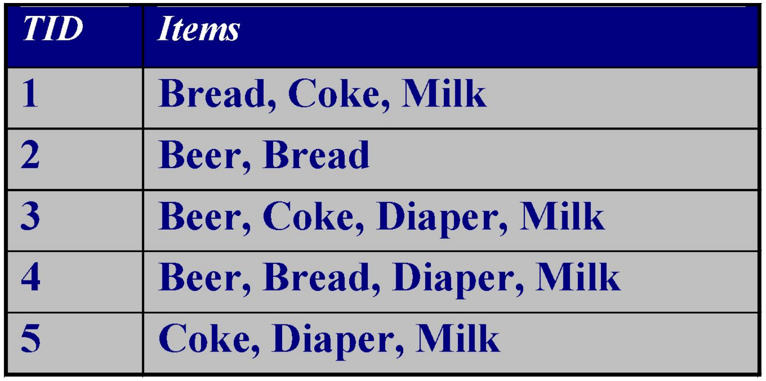
- **Frequent itemsets:** {m}, {c}, {b}, {j}, {m,b} , {b,c} , {c,j}.

Association Rules

- **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 this association rule is the probability of j given $I = \{i_1,\ldots,i_k\}$

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

Example of calculating support and confidence



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

- {m, br}
  - **Support** = 2/5 = 0.4
- {m, c}
  - **Support** = 3/5 = 0.6
- {m} →c
  - **Confidence** = 3/4 = 0.75
- {m, d} →be
  - **Confidence** = 2/3 = 0.66

Interesting Association Rules

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

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

  - Interesting rules are those with high positive or negative interest values (usually above 0.5)

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} →c**
  - **Confidence** = 2/4 = 0.5
  - **Interest** = |0.5 – 5/8| = 1/8
    - Item ***c*** appears in 5/8 of the baskets
    - Rule is not very interesting!

Finding Association Rules

- **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 on the left side
- **Hard part: Finding the frequent itemsets!**
  - If $\{i_1, i_2, \ldots, i_k\} \to j$ 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"

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

Mining Association Rules

- **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***→***C,D***) = support(**A,B,C,D**) / support(**A,B**)
    - **Variant 2:**
      - **Observation:** If **A,B,C**→**D** is below confidence, so is **A,B**→**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 $c = 0.75$
- 1) Frequent itemsets:
  - {b,m} {b,c} {c,m} {c,j} {m,c,b}
- 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: $c=3/6$~~

Compacting the Output

- **To reduce the number of rules we can post-process them and only output:**
  - **Maximal frequent itemsets:**
    No immediate superset is frequent
    - Gives more pruning

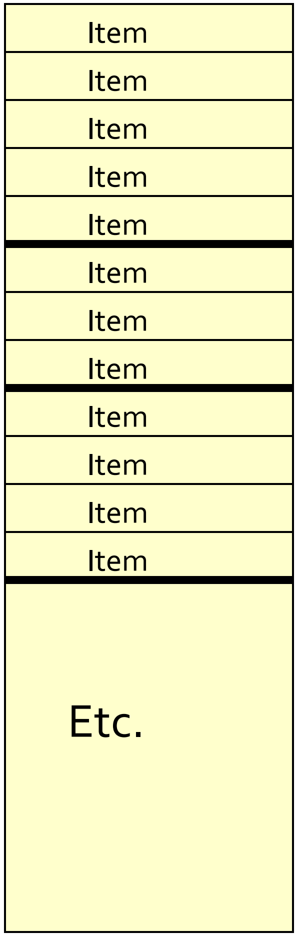
or

  - **Closed itemsets:**
    No immediate superset has the same count (> 0)
    - Stores not only frequent information, but exact counts

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 generate 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

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 (why?)

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**
- **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

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**
  - **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: $10^5(10^5\text{-}1)/2 = 5*10^9$
    - Therefore, $2*10^{10}$ (20 gigabytes) of memory needed

Counting Pairs in Memory

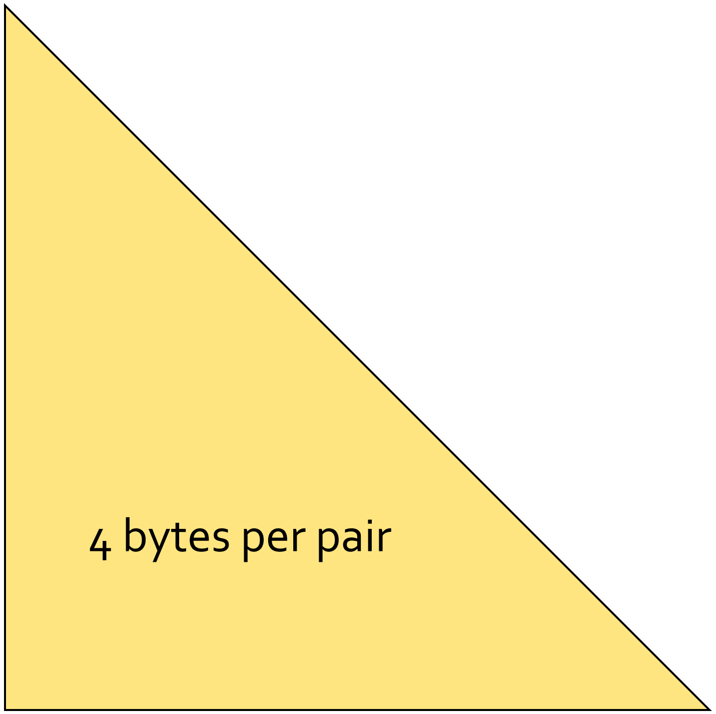
Two approaches:

- **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

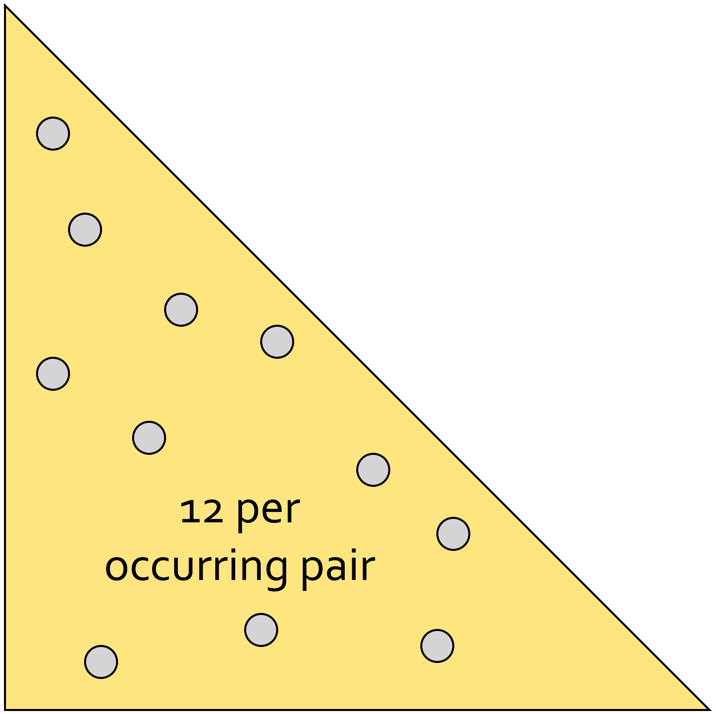
Note:

- **Approach 1** only requires 4 bytes per pair
- **Approach 2** uses 12 bytes per pair (but only for pairs with count > 0)

Comparing the 2 Approaches



Triangular Matrix



Triples

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 $(i-1)(n-i/2)+j-1$
  - Total number of pairs $\boldsymbol{n(n-1)/2}$; total bytes= $\boldsymbol{2n^2}$
  - **Triangular Matrix** requires 4 bytes per pair
- **Approach 2** uses **12 *bytes*** per occurring pair *(but only for pairs with count > 0)*
  - 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
  - P
  - T *2n*^{2}
  - **T**
- **Ap**

  (bu
  - Beats Approach 1 if less than 1/3 of possible pairs actually occur

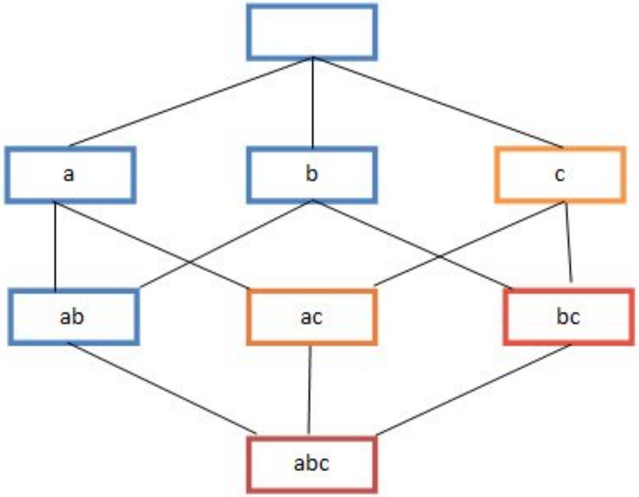
Problem is if we have too many items so the pairs do not fit into memory.

Can we do better?

A-Priori Algorithm

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?**

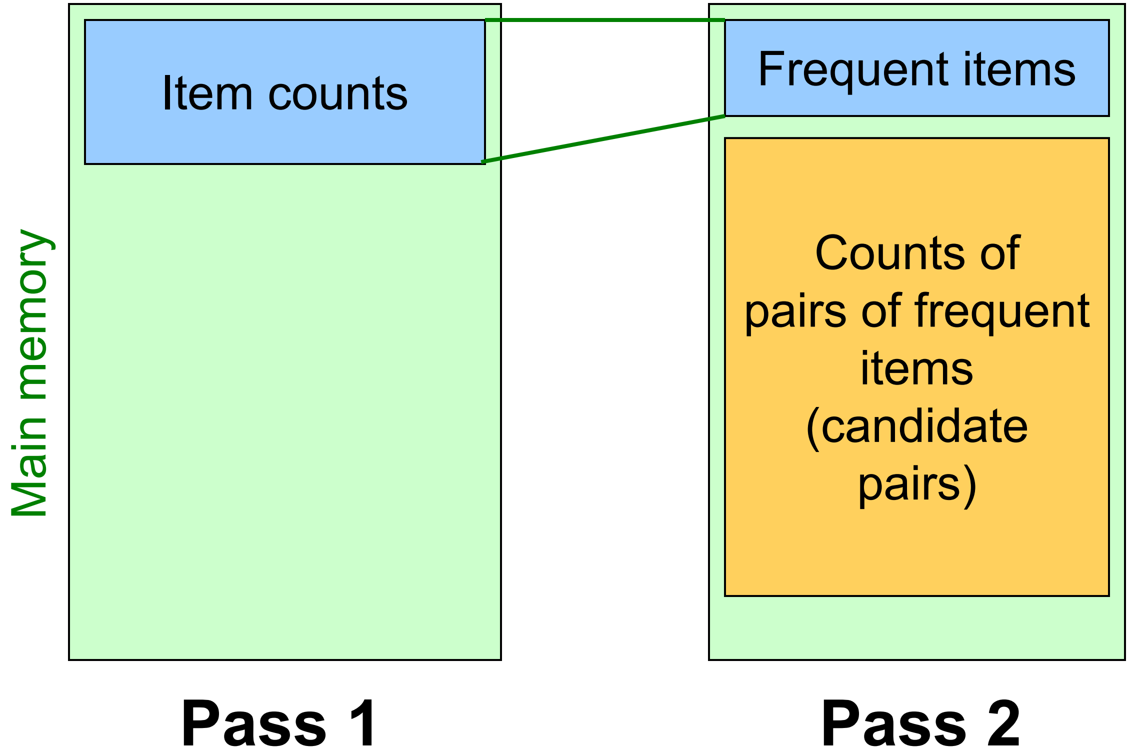
A-Priori Algorithm – (2)

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

- **Items that appear $\geq s$ times are the frequent items**

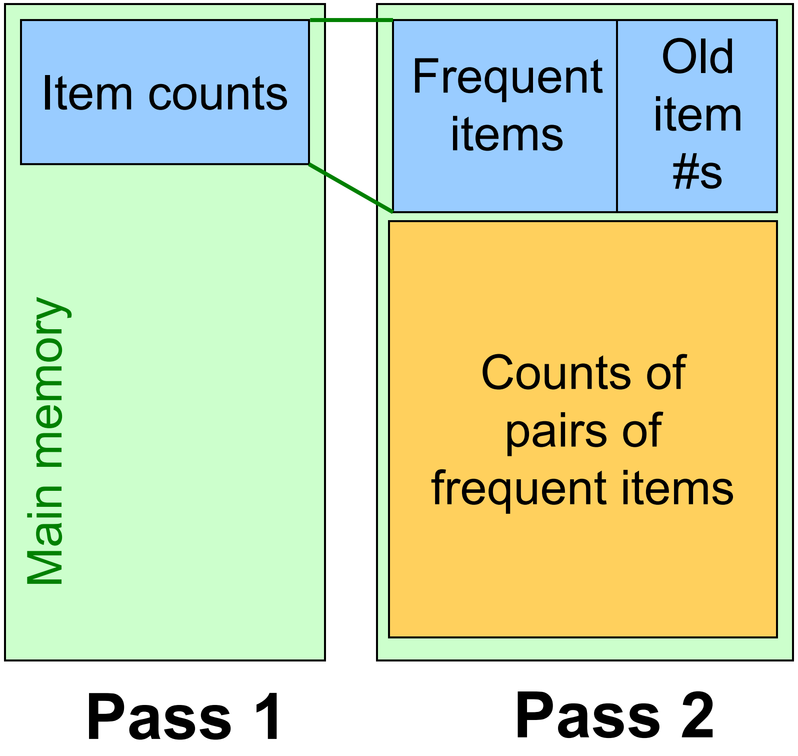
- **Pass 2:** Read baskets again and count in main memory only those pairs where both elements are frequent (from Pass 1)
  - Requires memory proportional to square of **frequent** items only (for counts)
  - Plus a list of the frequent items (so you know what must be counted)

Main-Memory: Picture of A-Priori



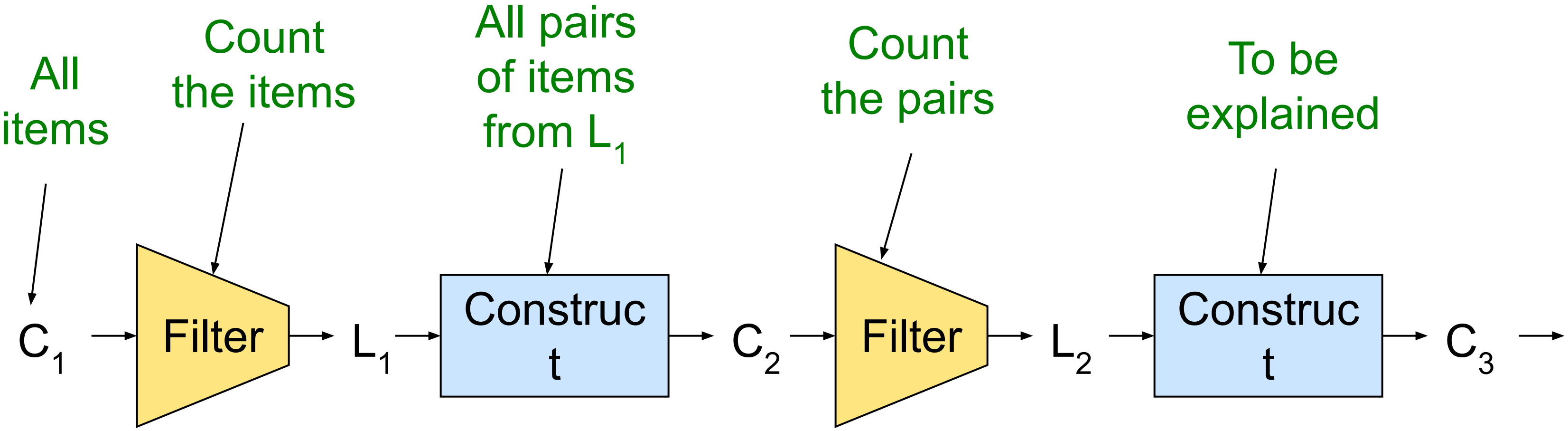
Detail for A-Priori

- You can use the triangular matrix method with ***n*** = number of frequent items
  - May save space compared with storing triples
- **Trick:** re-number frequent items 1,2,… and keep a table relating new numbers to original item numbers



Frequent Triples, Etc.

- **For each *k*, we construct two sets of *k-tuples*** (sets of size *k*):
  - ***C*$_k$** = ***candidate k-tuples*** = those that might be frequent sets (support **≥ s**) based on information from the pass for ***k*–1**
  - ***L*$_k$** = the set of truly frequent ***k***-tuples

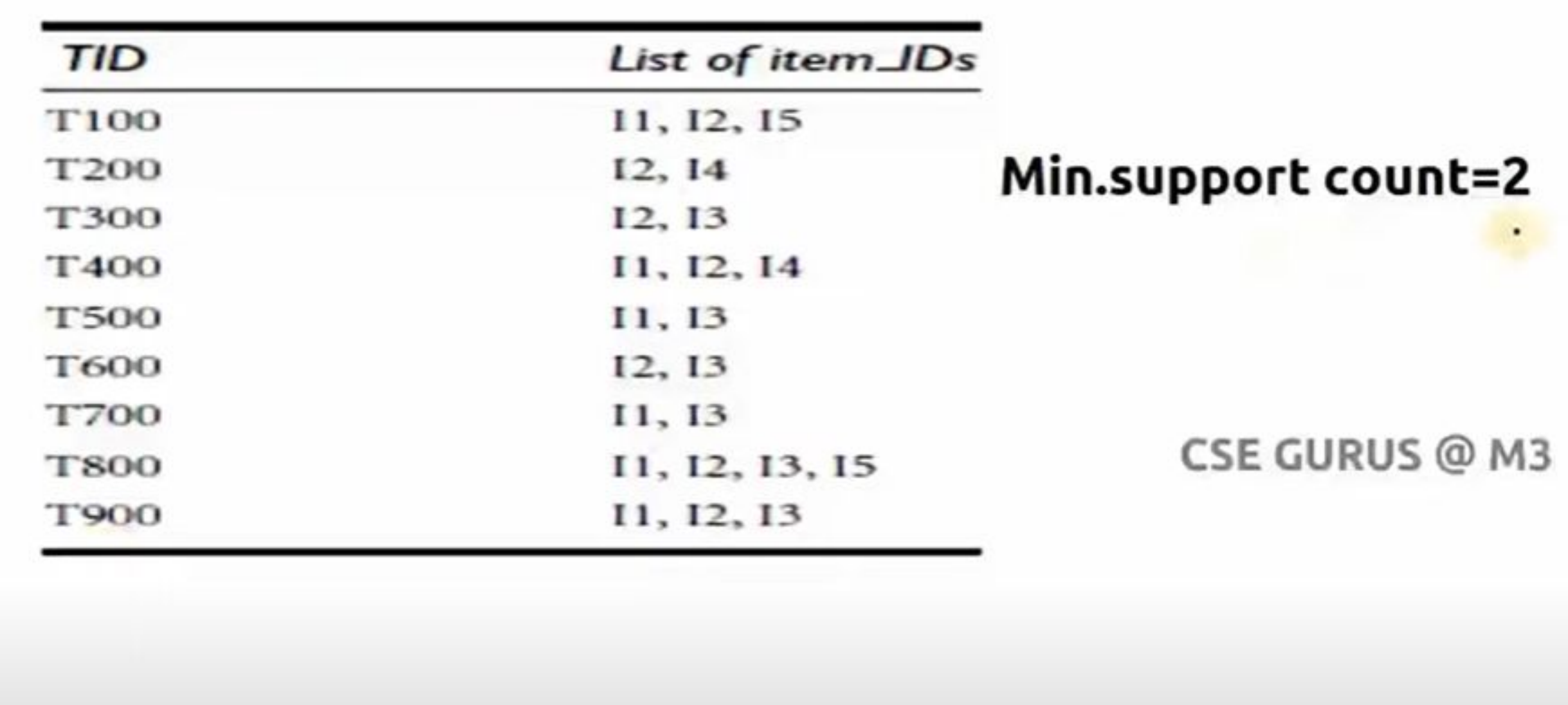


Example

** Note here we generate new candidates by generating C_k from L_{k-1} and L_1.
But that 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: 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} {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} }

Example

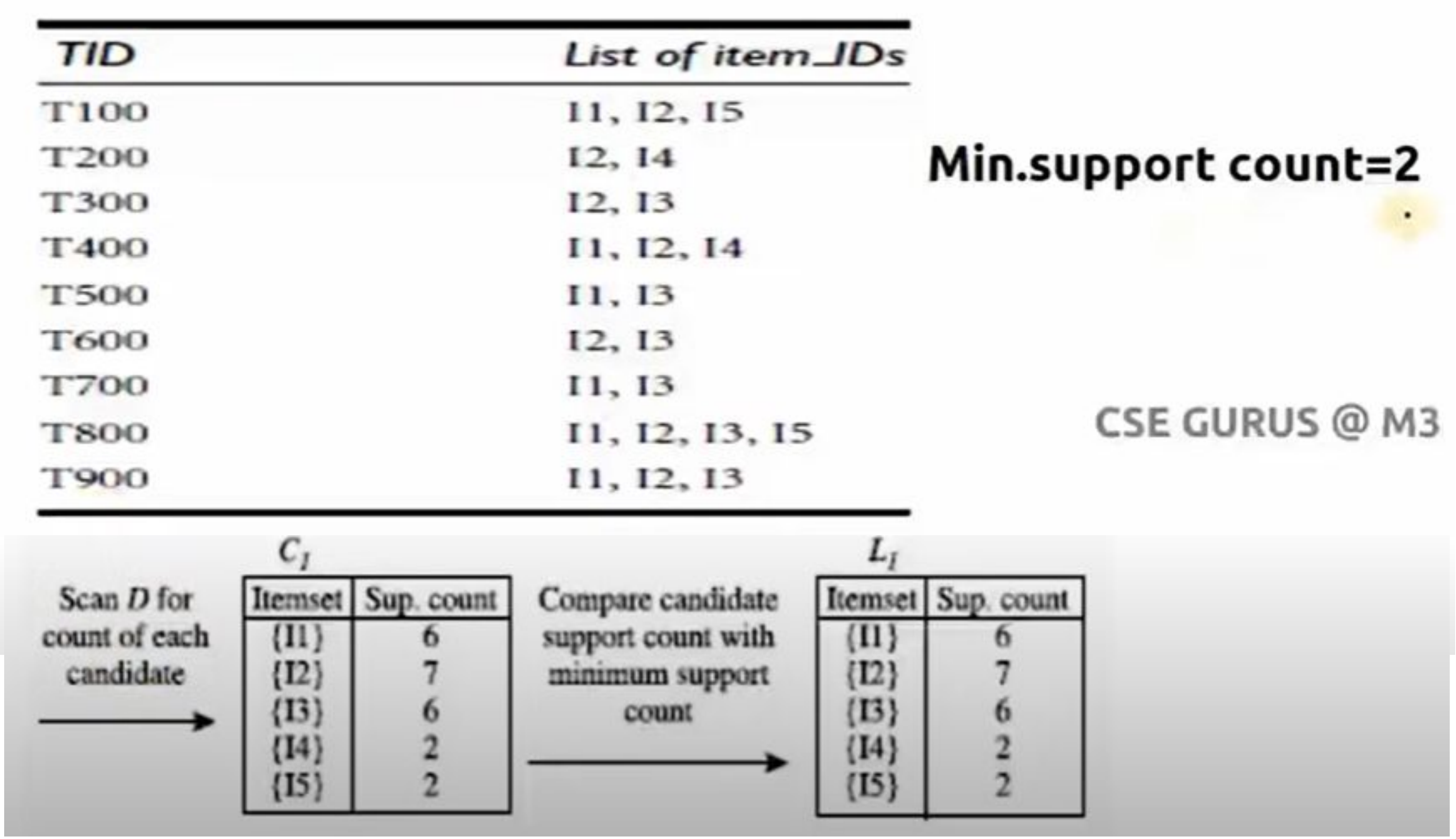


TID	List of item_IDs
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

Min.support count=2

CSE GURUS @ M3

Example



TID	List of item_IDs
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

Min.support count=2

CSE GURUS @ M3

Scan D for count of each candidate →

C_1

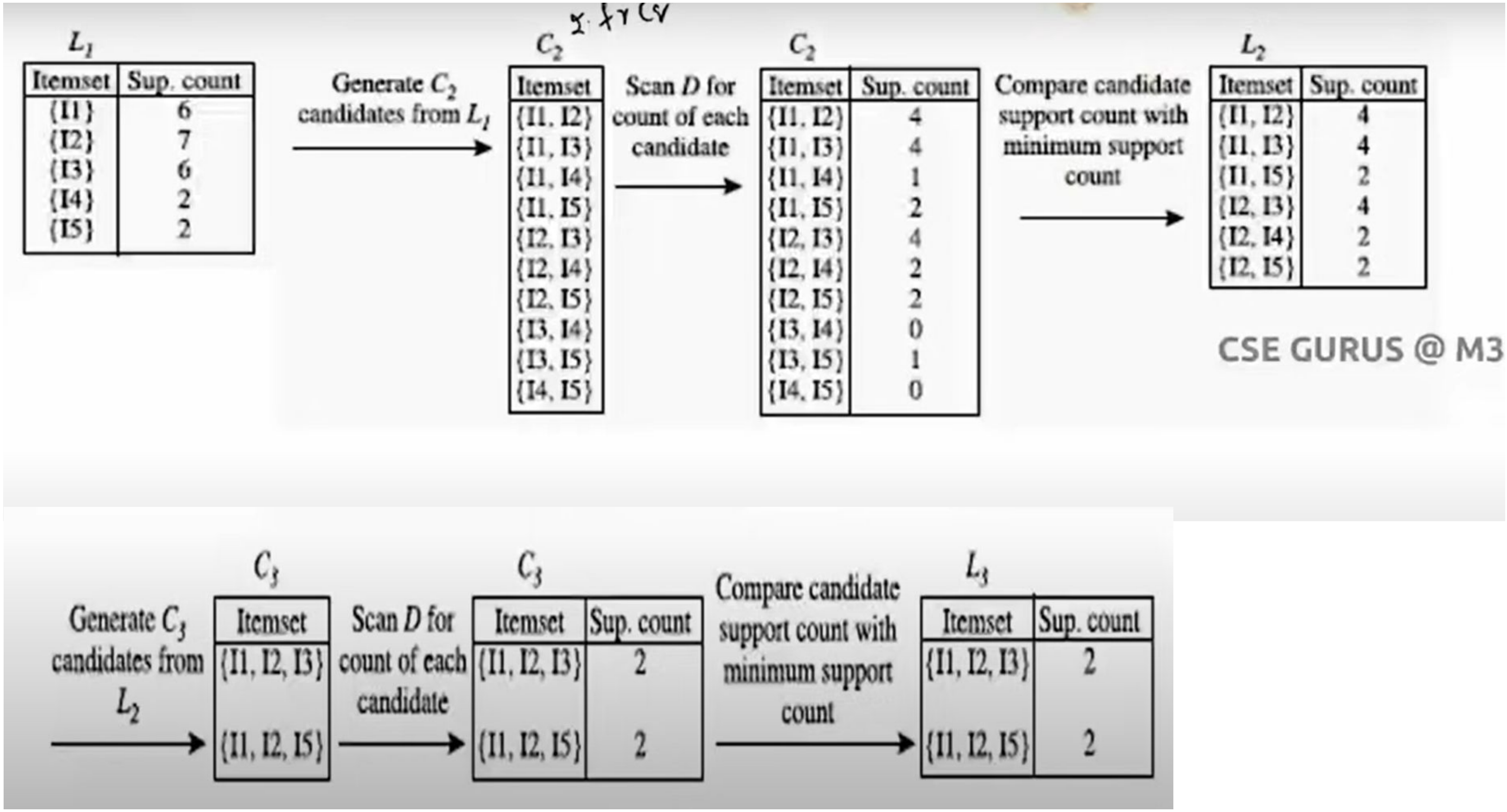
Itemset	Sup. count
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2

Compare candidate support count with minimum support count →

L_1

Itemset	Sup. count
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2

Example



L₁

Itemset	Sup. count
{I1}	6
{I2}	7
{I3}	6
{I4}	2
{I5}	2

Generate C_2 candidates from L_1 →

C_2 ৫·৪২ CV

Itemset
{I1, I2}
{I1, I3}
{I1, I4}
{I1, I5}
{I2, I3}
{I2, I4}
{I2, I5}
{I3, I4}
{I3, I5}
{I4, I5}

Scan D for count of each candidate →

C_2

Itemset	Sup. count
{I1, I2}	4
{I1, I3}	4
{I1, I4}	1
{I1, I5}	2
{I2, I3}	4
{I2, I4}	2
{I2, I5}	2
{I3, I4}	0
{I3, I5}	1
{I4, I5}	0

Compare candidate support count with minimum support count →

L_2

Itemset	Sup. count
{I1, I2}	4
{I1, I3}	4
{I1, I5}	2
{I2, I3}	4
{I2, I4}	2
{I2, I5}	2

CSE GURUS @ M3

Generate C_3 candidates from L_2 →

C_3

Itemset
{I1, I2, I3}
{I1, I2, I5}

Scan D for count of each candidate →

C_3

Itemset	Sup. count
{I1, I2, I3}	2
{I1, I2, I5}	2

Compare candidate support count with minimum support count →

L_3

Itemset	Sup. count
{I1, I2, I3}	2
{I1, I2, I5}	2

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

Additional Resources

- https://www.youtube.com/watch?v=h_l3b2CIQ_o