

Computational model for data mining

ANISHA JOSEPH

Market-Basket Data

- A large set of **items**, e.g., things sold in a supermarket.
- A large set of **baskets**, each of which is a small set of the items, e.g., the things one customer buys on one day.

Market-Baskets – (2)

- Really, a general many-to-many mapping (association) between two kinds of things, where the one (the **baskets**) is a set of the other (the **items**)
 - But we ask about connections among “items,” not “baskets.”
- The technology focuses on **common events**, not rare events (“long tail”).

Frequent Itemsets

- Given a set of transactions, find combinations of items (itemsets) that occur frequently

Support $s(I)$: number of transactions that contain itemset I

Market-Basket transactions

Items: {Bread, Milk, Diaper, Beer, Eggs, Coke}

| 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 |

Examples of frequent itemsets $s(I) \geq 3$

{Bread}: 4
{Milk} : 4
{Diaper} : 4
{Beer}: 3
{Diaper, Beer} : 3
{Milk, Bread} : 3

Applications – (1)

- **Items** = products; **baskets** = sets of products someone bought in one trip to the store.
- **Example application**: given that many people buy beer and diapers together:
 - Run a sale on diapers; raise price of beer.
- Only useful if many buy diapers & beer.

Applications – (2)

- **Baskets** = Web pages; **items** = words.
- **Example application:** Unusual words appearing together in a large number of documents, e.g., “Brad” and “Angelina,” may indicate an interesting relationship.

Applications – (3)

- **Baskets** = sentences; **items** = documents containing those sentences.
- **Example application:** Items that appear together too often could represent plagiarism.
- Notice items do not have to be “in” baskets.

Definition: Frequent Itemset

- Itemset

- A collection of one or more items
 - Example: {Milk, Bread, Diaper}
- k-itemset
 - An itemset that contains k items

- Support (σ)

- Count: Frequency of occurrence of an itemset
- E.g. $\sigma(\{\text{Milk, Bread, Diaper}\}) = 2$
- Fraction: Fraction of transactions that contain an itemset
- E.g. $s(\{\text{Milk, Bread, Diaper}\}) = 40\%$

- Frequent Itemset

- An itemset whose support is greater than or equal to a **minsup** threshold

| 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 |

$$s(I) \geq \text{minsup}$$

Mining Frequent Itemsets task

- **Input:** A set of transactions T , over a set of items I
- **Output:** All itemsets with items in I having
 - support \geq *minsup* threshold
- Problem parameters:
 - $N = |T|$: number of transactions
 - $d = |I|$: number of (distinct) items
 - w : max width of a transaction
 - Number of possible itemsets? $M = 2^d$
- Scale of the problem:
 - WalMart sells 100,000 items and can store billions of baskets.
 - The Web has billions of words and many billions of pages.

Computation Model

- Typically, data is kept in flat files rather than in a database system.
 - Stored on disk.
 - Stored basket-by-basket.
 - Expand baskets into pairs, triples, etc. as you read baskets.
 - Use k nested loops to generate all sets of size k .

Example file: retail

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
30 31 32
33 34 35
36 37 38 39 40 41 42 43 44 45 46
47 48 49 50
51 52 53 54 55 56 57 58
59 60 61 62 63 64 65
66 67 68
69 70 71 72
73 74 75 76 77 78 79
80 81 82 83 84 85 86 87
88 89 90
91 92 93 94 95
96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118
119 120 121 122 123 124 125 126 127 128 129 130 131 132 133
134 135 136
137 138 139 140 141 142 143 144 145 146 147 148 149
150 151 152
153 154 155 156 157 158 159
```

Example: items are positive integers, and each basket corresponds to a line in the file of space separated integers

Computation Model – (2)

- The true cost of mining disk-resident data is usually the **number of disk I/O's**.
- In practice, association-rule algorithms read the data in **passes** – all baskets read in turn.
- Thus, we measure the cost by the **number of passes** an algorithm takes.

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.
 - The number of different things we can count is limited by main memory.
 - Swapping counts in/out is a disaster (*why?*).

The Apriori Principle

- **Apriori** principle (Main observation):
 - If an itemset is **frequent**, then all of its **subsets** must also be frequent
 - If an itemset is **not frequent**, then all of its **supersets** cannot be frequent

$$\forall X, Y : (X \subseteq Y) \Rightarrow s(X) \geq s(Y)$$

- The support of an itemset **never exceeds** the support of its subsets
- This is known as the **anti-monotone** property of support

Association Rule Mining

- Given a set of transactions, find **rules** that will predict the occurrence of an item based on the occurrences of other items in the transaction

Market-Basket transactions

| 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 of Association Rules

$\{Diaper\} \rightarrow \{Beer\},$
 $\{Milk, Bread\} \rightarrow \{Eggs, Coke\},$
 $\{Beer, Bread\} \rightarrow \{Milk\},$

Implication means **co-occurrence**,
not causality!

Definition: Association Rule

- **Association Rule**

- An implication expression of the form $X \rightarrow Y$, where X and Y are itemsets
- Example:
 $\{Milk, Diaper\} \rightarrow \{Beer\}$

- **Rule Evaluation Metrics**

- **Support** (s)
 - Fraction of transactions that contain both X and Y
 - the probability $P(X, Y)$ that X and Y occur together
- **Confidence** (c)
 - Measures how often items in Y appear in transactions that contain X
 - the conditional probability $P(Y|X)$ that Y occurs given that X has occurred.

| 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:

$\{Milk, Diaper\} \Rightarrow Beer$

$$s = \frac{\sigma(Milk, Diaper, Beer)}{|T|} = \frac{2}{5} = 0.4$$

$$c = \frac{\sigma(Milk, Diaper, Beer)}{\sigma(Milk, Diaper)} = \frac{2}{3} = 0.67$$

Association Rule Mining Task

- **Input:** A set of transactions T , over a set of items I
- **Output:** All rules with items in I having
 - support $\geq \text{minsup}$ threshold
 - confidence $\geq \text{minconf}$ threshold

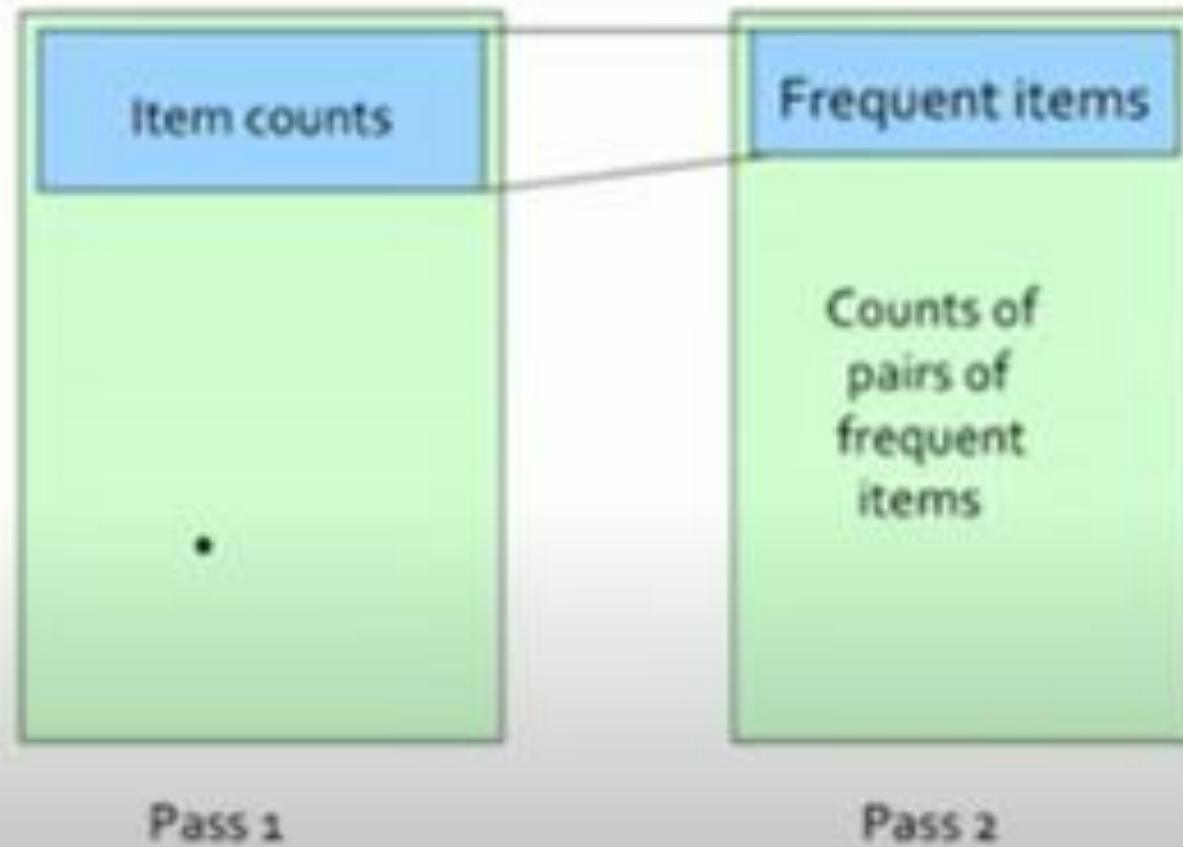
A-Priori Algorithm

- A two-pass approach called *a-priori* limits the need for main memory.
- Key idea: *monotonicity*: if a set of items appears at least s times, so does every subset of S .
- **Contrapositive for pairs**: if item i does not appear in s baskets, then no pair including i can appear in s baskets.

A-Priori Algorithm – (2)

- **Pass 1:** Read baskets and count in main memory the occurrences of each item.
 - Requires only memory proportional to #items.
- Items that appear at least s times are the *frequent 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 (for counts), plus a list of the frequent items (so you know what must be counted).

Picture of A-Priori



a) Hash Based Technique

- Hash table is used as data structure
- First iteration is required to count support of each itemset.
- From 2nd iteration, efforts are made to enhance execution of Apriori by utilizing hash table concept.
- Hash table minimizes the number of itemset generated in the second iteration.
- In second iteration, ie 2-itemset generation, for every combination of two items, we map them into the diverse bucket of hash table structure and increment the bucket count.
- If count of bucket is less than min.sup.count, we remove them from candidate sets.

a) Hash Based Technique...

Data Base

| TID | List of Items |
|-----|---------------|
| T1 | I1,I2,I5 |
| T2 | I2,I4 |
| T3 | I2,I3 |
| T4 | I1,I2,I4 |
| T5 | I1,I3 |
| T6 | I2,I3 |
| T7 | I1,I3 |
| T8 | I1,I2,I3,I5 |
| T9 | I1,I2,I3 |

C1

| Itemset | Support Count |
|---------|---------------|
| I1 | 6 |
| I2 | 7 |
| I3 | 6 |
| I4 | 2 |
| I5 | 2 |

Hash Function

| Itemset | Count | Hash Function |
|---------|-------|-------------------------|
| I1,I2 | 4 | $[1*10+2]\text{mod}7=5$ |
| I1,I3 | 4 | $[1*10+3]\text{mod}7=6$ |
| I1,I4 | 1 | $[1*10+4]\text{mod}7=0$ |
| I1,I5 | 2 | $[1*10+5]\text{mod}7=1$ |
| I2,I3 | 4 | $[2*10+3]\text{mod}7=2$ |
| I2,I4 | 2 | $[2*10+4]\text{mod}7=3$ |
| I2,I5 | 2 | $[2*10+5]\text{mod}7=4$ |
| I3,I4 | 0 | |
| I3,I5 | 1 | $[3*10+5]\text{mod}7=0$ |
| I4,I5 | 0 | |

Min.Support Count=3

Order of items I1=1, I2=2, I3=3, I4=4, I5=5

$$H(x,y) = ((\text{Order of first}) * 10 + (\text{Order of second})) \text{mod} 7$$

Hash Based Technique...

Hash table structure to generate L2

| Bucket Address | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|----------------|------------------------|-----------|-----------|-----------|-----------|-----------|-----------|
| Bucket Count | 2 | 2 | 4 | 2 | 2 | 4 | 4 |
| Bucket Content | {I1-I4}-1 {I3-I5}-1 | {I1-I5}-2 | {I2-I3}-4 | {I2-I4}-2 | {I2-I5}-2 | {I1-I2}-4 | {I1-I3}-4 |
| L2 | No | No | Yes | No | No | Yes | Yes |

Advantages:

1. Reduce the number of scans
2. Remove the large candidate that cause high input/output cost

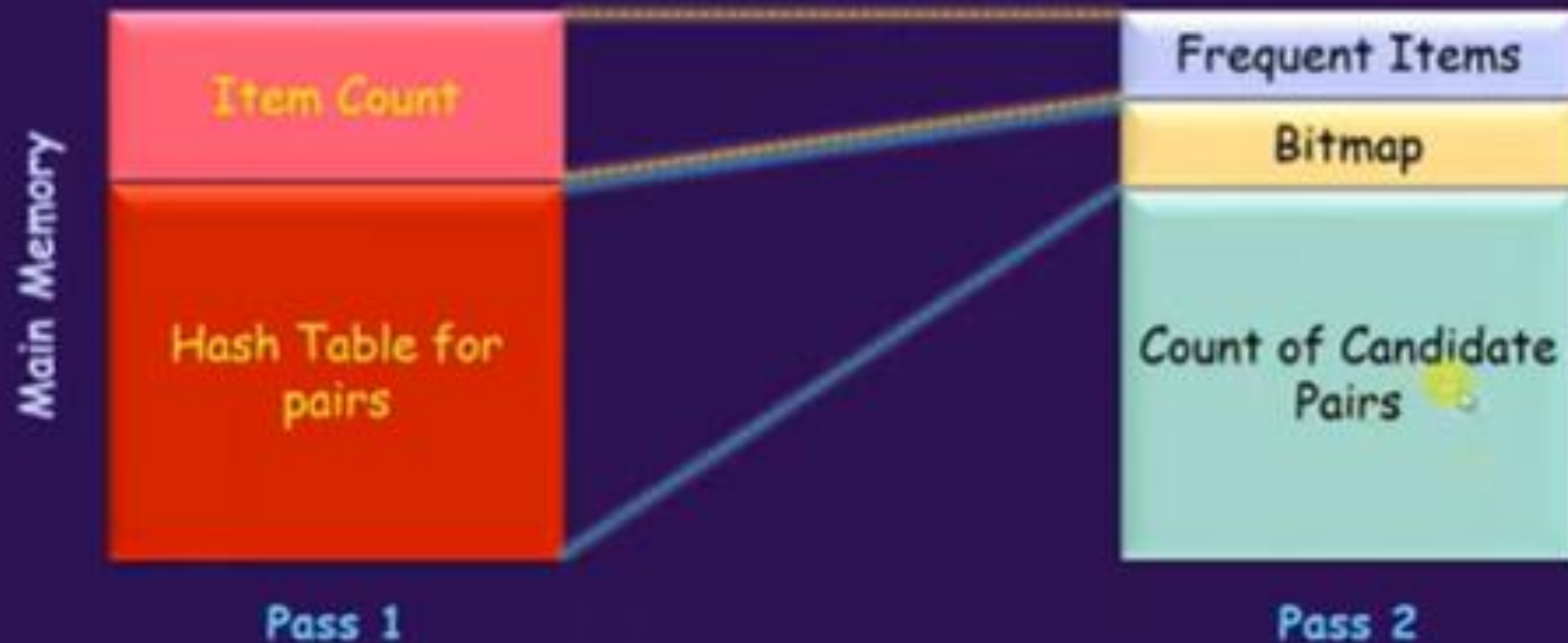
PCY (Park-Chen-Yu) Algorithm

Overview

- Developed to efficiently find frequent itemsets in large datasets
- Algorithms like Apriori were computationally expensive for with large datasets.
- This was due to need to generate and test a large number of candidate itemsets.
- Uses a hash-based approach to count itemset frequency and prune infrequent itemsets.
- This avoids the need to generate candidate itemsets, making the algorithm more efficient.
- It is commonly used in data mining and association rule learning.
- Applications of the PCY algorithm include market basket analysis, recommendation systems, and web log analysis.

PCY (Park-Chen-Yu) Algorithm

Main Memory: Picture of PCY



PCY (Park-Chen-Yu) Algorithm

Algorithm

Pass 1:

FOR (each basket):

FOR (each item in the basket):

add 1 to item's count

FOR (each pair of items):

hash the pair to a bucket

add 1 to the count for that bucket

Pass 2:

Count all pairs $\{i, j\}$ that meet the conditions for being a candidate pair:

1. Both i and j are frequent items

2. The pair $\{i, j\}$ hashes to a bucket whose bit in the bit vector is 1

PCY (Park-Chen-Yu) Algorithm

Example

Find frequent item pairs using PCY algorithm considering threshold as 2

| Transactions | Items |
|--------------|---------|
| T1 | 1, 2, 3 |
| T2 | 4, 5 |
| T3 | 1, 4, 5 |
| T4 | 1, 2, 4 |
| T5 | 3, 4, 5 |
| T6 | 2, 4, 5 |

PCY (Park-Chen-Yu) Algorithm

Example

Step 1: Find length of each item and eliminate items whose count is less than 1

| Transactions | Items |
|--------------|---------|
| T1 | 1, 2, 3 |
| T2 | 4, 5 |
| T3 | 1, 4, 5 |
| T4 | 1, 2, 4 |
| T5 | 3, 4, 5 |
| T6 | 2, 4, 5 |

| | |
|---|---|
| 1 | 3 |
| 2 | 3 |
| 3 | 2 |
| 4 | 5 |
| 5 | 4 |

Eliminate those items whose count is less than 1, but in our case no item count is less than 1.

Candidate Item-set:
{1, 2, 3, 4, 5}

PCY (Park-Chen-Yu) Algorithm

Example

Step 2: Form key-value pair where key is the item-pair and value is their occurrence

| Transactions | Items | Pairs with their counts |
|--------------|---------|---|
| T1 | 1, 2, 3 | $\{(1,2) : 2\}, \{(2,3) : 1\}, \{(1,3) : 1\}$ |
| T2 | 4, 5 | $\{(4,5) : 4\}$ |
| T3 | 1, 4, 5 | $\{(1,4) : 2\}, \{(1,5) : 1\}$ |
| T4 | 1, 2, 4 | $\{(2,4) : 2\}$ |
| T5 | 3, 4, 5 | $\{(3,4) : 1\}, \{(3,5) : 1\}$ |
| T6 | 2, 4, 5 | $\{(2,5) : 1\}$ |

PCY (Park-Chen-Yu) Algorithm

Example

Step 3: Eliminate pairs whose count of occurrence does not satisfy threshold value

| Transactions | Pairs |
|--------------|---|
| T1 | $\{(1,2) : 2\}, \{(2,3) : 1\}, \{(1,3) : 1\}$ |
| T2 | $\{(4,5) : 4\}$ |
| T3 | $\{(1,4) : 2\}, \{(1,5) : 1\}$ |
| T4 | $\{(2,4) : 2\}$ |
| T5 | $\{(3,4) : 1\}, \{(3,5) : 1\}$ |
| T6 | $\{(2,5) : 1\}$ |

Since, threshold is 2, hence eliminate those pairs whose count is less than 2

Final Pairs:
 $\{(1,2), (4,5), (1,4), (2,4)\}$

PCY (Park-Chen-Yu) Algorithm

Example

Step 4: Find bucket numbers

Final Pairs: $\{(1,2), (4,5), (1,4), (2,4)\}$

Now, Apply hash function to find bucket no. :

For pair (i,j) , hash function will be $(i * j) \bmod 10$

| Pair | Bucket No. |
|-------|------------------------|
| (1,2) | $(1 * 2) \bmod 10 = 2$ |
| (4,5) | $(4 * 5) \bmod 10 = 0$ |
| (1,4) | $(1 * 4) \bmod 10 = 4$ |
| (2,4) | $(2 * 4) \bmod 10 = 8$ |

PCY (Park-Chen-Yu) Algorithm

Example

Step 5: Create Candidate set table

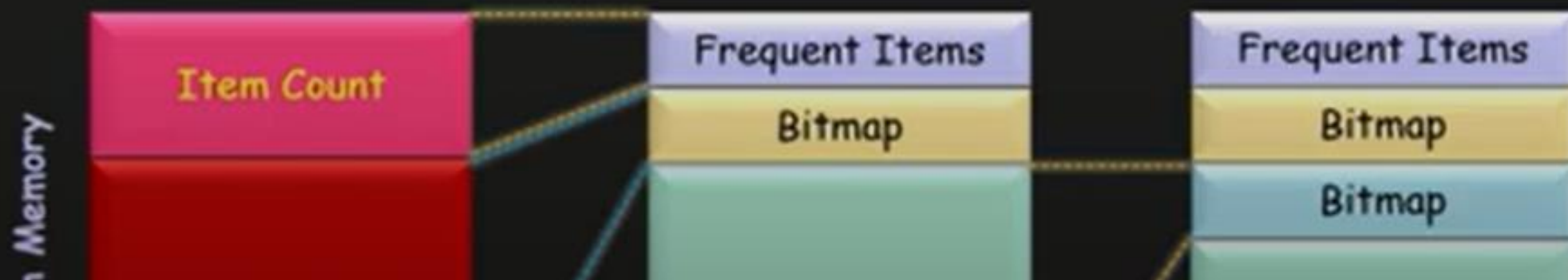
| Bit Vector | Bucket No | Count | Pairs | Candidate Set |
|------------|-----------|-------|-------|---------------|
| 1 | 2 | 2 | (1,2) | (1,2) |
| 1 | 0 | 4 | (4,5) | (4,5) |
| 1 | 4 | 2 | (1,4) | (1,4) |
| 1 | 8 | 2 | (2,4) | (2,4) |

Therefore, Candidate pair set: $\{(1,2), (4,5), (1,4), (2,4)\}$

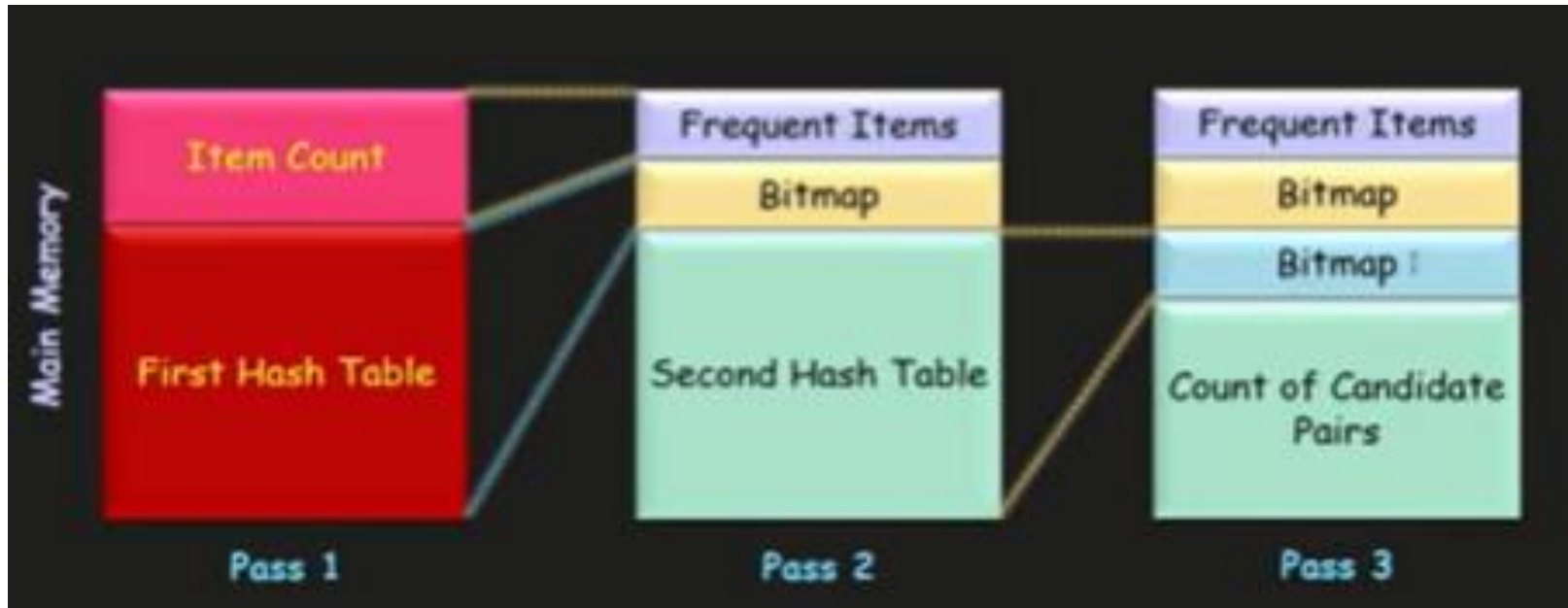
Multistage PCY - Refinement of PCY algorithm

Why use Multistage PCY?

1. More Hash Tables: Multistage uses several successive hash tables to further reduce the number of candidate pairs.
2. Additional Passes: Unlike PCY, which takes two passes to find frequent pairs, Multistage takes more than two passes. This allows it to refine the candidate pairs more effectively.
- 3. Memory Utilization: In Multistage, bit-vectors eventually consume all of the main memory. This is a trade-off for the increased accuracy in candidate pair selection.



The Multistage Algorithm



The Multistage Algorithm

Algorithm

I

1. Pass 1: Count items and Hash pairs $\{i, j\}$
2. Pass 2:
if (i, j) are frequent) and $(\{i, j\})$ hashes to frequent bucket in B1):
Hash pairs $\{i, j\}$ into Hash2
3. Pass 3:
if (i, j) are frequent) and $(\{i, j\})$ hashes to freq. bucket in B1) and $(\{i, j\})$ hashes to freq. bucket in B2):
Count pairs $\{i, j\}$

Example

Basket 1: {apple, banana, mango} \rightarrow {A, B, M}

Basket 2: {apple, orange} \rightarrow {A, O}

Basket 3: {banana, mango} \rightarrow {B, M}

Basket 4: {apple, banana} \rightarrow {A, B}

Basket 5: {mango, orange} \rightarrow {M, O}

Minimum support count to be 2.

$$\begin{array}{l} 1 \leftarrow A \rightarrow 3 \\ 2 \leftarrow B \rightarrow 3 \\ 3 \leftarrow M \rightarrow 3 \\ 4 \leftarrow O \rightarrow 2 \end{array} \quad \left. \vphantom{\begin{array}{l} 1 \\ 2 \\ 3 \\ 4 \end{array}} \right\}$$

$$\text{Hash 1} = (i + j) \bmod 3$$

Ques-1

$$(A, B) \rightarrow (1, 2) \rightarrow (1 + 2) \bmod 3 = 0$$

$$(B, m) \rightarrow (2, 3) \rightarrow (2 + 3) \bmod 3 = 2$$

$$(A, m) \rightarrow (1, 3) \rightarrow (1 + 3) \bmod 3 = \underline{1}$$

$$(A, 0) \rightarrow (1, 4) \rightarrow (1 + 4) \bmod 3 = 2$$

$$(m, 0) \rightarrow (3, 4) \rightarrow (3 + 4) \bmod 3 = \underline{1}$$

Bucket 0 \rightarrow 1 pair

Bucket 1 \rightarrow 2 pairs

Bucket 2 \rightarrow 2 pairs

Frequent
Buckets
0, 1, 2

Pass 2:

$$\text{Hash 2} = (i * j) \bmod 3$$

$$(B, m) \rightarrow (2, 3) = (2 \times 3) \bmod 3 = 0$$

$$(A, m) \rightarrow (1, 3) = (1 \times 3) \bmod 3 = 0$$

$$(A, 0) \rightarrow (1, 4) = (1 \times 4) \bmod 3 = 1$$

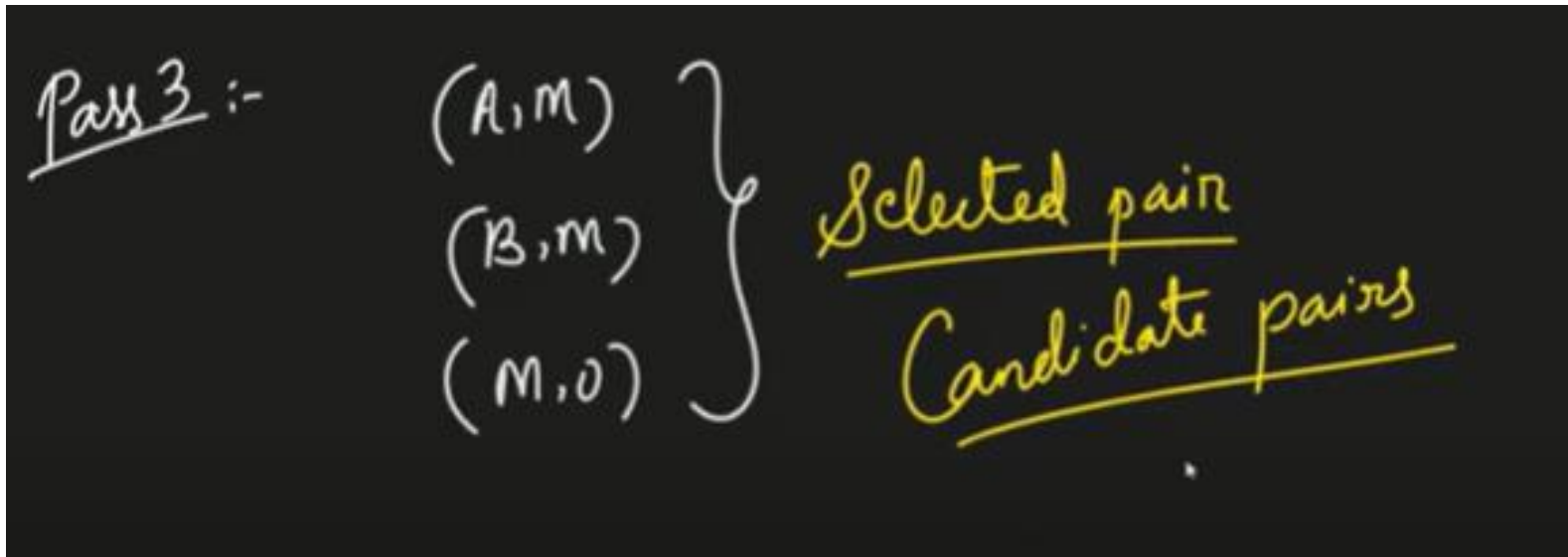
$$(M, 0) \rightarrow (3, 4) = (3 \times 4) \bmod 3 = 0$$

Bucket 0 \rightarrow (A, m), (B, m) and (m, 0) \rightarrow 3 pairs ✓

Bucket 1 \rightarrow (A, 0) \rightarrow 1 pair ✗

The Multistage Algorithm

Candidate pairs satisfied in both passes.



The Algorithm of Savasere, Omiecinski, and Navathe(SON Algorithm)

Overview

- SON stands for Savasere, Omiecinski, and Navathe.
- Improvement of PCY algorithm and its versions.
- Used to find frequent itemsets in large datasets.
- Uses two MapReduce passes.
- Can be parallelized for faster processing.
- Used in data mining and market basket analysis.

SON Algorithm and MapReduce

Ideology

Pass 1: Finds candidate itemsets

Pass 2: Finds correct frequent itemsets



SON Algorithm and MapReduce

Algorithm

Pass 1: First Map task

1. Divide data into chunks
2. For each chunk the support will be:
$$\text{support} = s / \text{number of chunks}$$
3. Using random algorithm find frequent itemset in that chunk and output them as $(F, 1)$ where F is frequent itemset and 1 is irrelevant number ●

SON Algorithm and MapReduce

Algorithm

Pass 1: First Reduce task

1. The value is ignored from $(F, 1)$ and key part F is assigned to each reduce task
2. Produces keys that appear one or more times
3. These produced keys are candidate itemsets

SON Algorithm and MapReduce

Algorithm

Pass 2: Second Map task

1. Each Map task takes the output from first reduce task and a portion of the input data file
2. Every map task will have all the candidate itemsets
3. It counts the occurrence of each candidate itemset among the baskets in the portion of the dataset
4. Output of the map task is (C, v)

Where, C is candidate itemset

v is the support of that itemset among the baskets that were input to this map task

SON Algorithm and MapReduce

Algorithm

Pass 2: Second Reduce task

1. Each reduce task takes itemsets (keys from map phase)
2. Calculates sum of associated values (support) for each itemset
3. Checks the below condition:
 if support of itemset $\geq s$:
 emit(itemset)
4. Itemset that does not satisfies the support criteria are not transmitted as output

SON Algorithm and MapReduce

Advantage

False Negative

In reality the itemset
is frequent

As per the results the itemset
is not frequent

False Negatives will not be entertained




Toivonen's Algorithm

- Takes Single full pass over the dataset
- Pick a random sample
- To determine all possible association rules
- Verify the results with the rest
- Produces exact association rules in one full pass

Toivonen's Algorithm: Example



- Consider the set of items $I = \{A, B, C, D, E\}$ and let the combined frequent itemsets of size 1 to 3 be $S = \{\{A\}, \{B\}, \{C\}, \{D\}, \{AB\}, \{AC\}, \{BC\}, \{AD\}, \{CD\}, \{ABC\}\}$

- 1. The negative border is $\{\{E\}, \{BD\}, \{ACD\}\}$.
- 2. The set $\{E\}$ is the only 1-itemset not contained in S .
- 3. $\{BD\}$ is the only 2-itemset not in S but whose 1-itemset subsets are.
- 4. $\{ACD\}$ is the only 3-itemset whose 2-itemset subsets are all in S .

The negative border is important since it is necessary to determine the support

Toivonen algorithm

Suppose the items are $\{A,B,C,D,E\}$ and we have found the following itemsets to be frequent in the sample: $\{A\}$, $\{B\}$, $\{C\}$, $\{D\}$, $\{B,C\}$, $\{C,D\}$.

Thus, the negative border consists of five sets: $\{E\}$, $\{A,B\}$, $\{A,C\}$, $\{A,D\}$ and $\{B,D\}$.

6.4.7 Exercises for Section 6.4

Exercise 6.4.1: Suppose there are eight items, A, B, \dots, H , and the following are the maximal frequent itemsets: $\{A, B\}$, $\{B, C\}$, $\{A, C\}$, $\{A, D\}$, $\{E\}$, and $\{F\}$. Find the negative border.

Advantages and Disadvantages

- Advantages:
 - Reduced failure probability, while keeping candidate-count low enough for memory
- Disadvantages:
 - a small yet non-zero probability that it will fail to produce any answer at all.

Counting Frequent Items in a Stream



- The telecommunication network fault analysis, it is important to know what are the frequent itemsets happening before the faults on the network during the recent period of time.