COMP336 — Big Data

Week 10 Lecture 1: Frequent Itemsets

Diego Mollá

COMP336 2018H1

Abstract

We explore one of the major families of techniques for characterising data: The discovery of frequent itemsets. This problem is often viewed as the discovery of "association rules", although the latter is a more complex characterisation of data, whose discovery depends fundamentally on the discovery of frequent itemsets. We introduce the "market-basket" model of data, and tackle the problem of identifying frequent itemsets in several steps. First we will introduce the A-Priori algorithm and later we will focus on approaches that would work on very large data sets.

Update May 13, 2018

Contents

Reading

- Leskovec, Rajaraman, Ullman (2014): Mining of Massive Datasets, Chapter 6. `http://www.mmds.org/`

1 The Market-Basket Model

1.1 Association Rules

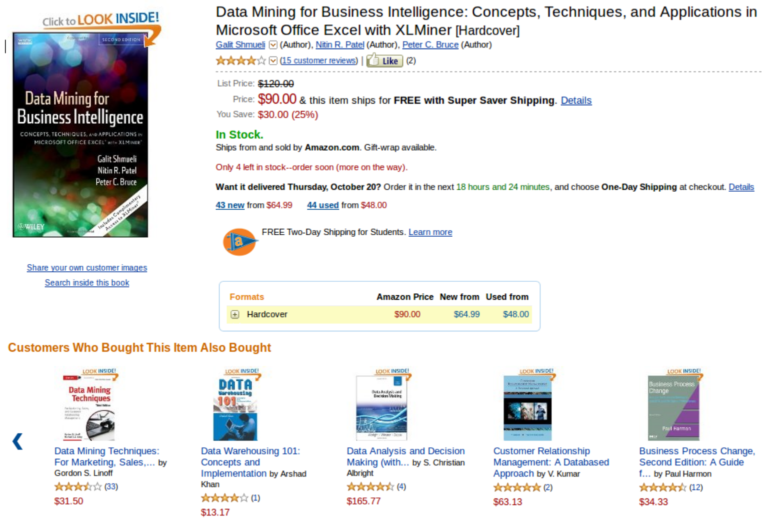
What Are Association Rules?

- Study of "what goes with what":
  - "Customers who bought X also bought Y".

  - What symptoms go with what diagnosis.

- Transaction-based or event-based.
- Also called "market basket analysis" and "affinity analysis".
- Originated with study of customer transactions databases to determine associations among items purchased.

Used in Many Recommender Systems


Click to LOOK INSIDE!

Data Mining for Business Intelligence: Concepts, Techniques, and Applications in Microsoft Office Excel with XLMiner [Hardcover]

Galit Shmueli (Author), Nitin R. Patel (Author), Peter C. Bruce (Author)

(15 customer reviews) | Like (2)

List Price: ~~$120.00~~

Price: $90.00 & this item ships for **FREE with Super Saver Shipping**. Details

You Save: $30.00 (25%)

In Stock.

Ships from and sold by **Amazon.com**. Gift-wrap available.

Only 4 left in stock--order soon (more on the way).

Want it delivered Thursday, October 20? Order it in the next 18 hours and 24 minutes, and choose **One-Day Shipping** at checkout. Details

43 new from $64.99   44 used from $48.00

FREE Two-Day Shipping for Students. Learn more

Share your own customer images

Search inside this book

Formats	Amazon Price	New from	Used from
Hardcover	$90.00	$64.99	$48.00

Customers Who Bought This Item Also Bought

Data Mining Techniques: For Marketing, Sales,... by Gordon S. Linoff (33) $31.50

Data Warehousing 101: Concepts and Implementation by Arshad Khan (1) $13.17

Data Analysis and Decision Making (with... by S. Christian Albright (4) $165.77

Customer Relationship Management: A Databased Approach by V. Kumar (2) $63.13

Business Process Change, Second Edition: A Guide f... by Paul Harmon (12) $34.33

A Working Example

Purchase of phone faceplates

A store that sells accessories for mobile phones runs a promotion on faceplates. Customers who purchase multiple faceplates from a choice of six different colours get a discount. The store managers, who would like to know what colours of faceplates customers are likely to purchase together, collected a transaction database.

Transaction	Faceplate colours purchased			
1	red	white	green	
2	white	orange		
3	white	blue		
4	red	white	orange	
5	red	blue		
6	white	blue		
7	white	orange		
8	red	white	blue	green
9	red	white	blue	
10	yellow			

Candidate Rules

Example of a rule

IF {red, white} THEN {green}

Terminology

Antecedent {red, white}.

Consequent {green}.

Item set {red, white, green, orange, blue, yellow}.

The antecedent and the consequent are *disjoint*:

- They have no items in common.

Many Rules are Possible
For example: Transaction 1 supports several rules, such as

1. "If red, then white" ("If a red faceplate is purchased, then so is a white one").
2. "If white, then red".
3. "If red and white, then green".
4. . . .

Frequent Item Sets

- Ideally, we want to create all possible combinations of items.
- *Problem*: computation time grows exponentially as # items increases.
- *Solution*: consider only "frequent item sets".
- Criterion for frequent: *support*.

1.2 Frequent Itemsets

Definition of Frequent Itemsets

- A set of items that appears in many baskets is said to be "frequent".
- Assume there is a number s, called the *support threshold*.
- If I is a set of items, the *support* for I is the number of baskets for which I is a subset.
- We then say that I is frequent if its support is s or more.

In our Working Example

The support for the item set {red, white} is 4

Transaction	Faceplate colours purchased			
1	*red*	*white*	green	
2	white	orange		
3	white	blue		
4	*red*	*white*	orange	
5	red	blue		
6	white	blue		
7	white	orange		
8	*red*	*white*	blue	green
9	*red*	*white*	blue	
10	yellow			

Applications of Frequent Itemsets

Brick-and-Mortar Retailing

Items: Products.

Baskets: Sets of products someone bought in one trip to the store.

- By finding frequent itemsets, a retailer can learn what is commonly bought together and arrange the marketing and sales strategy accordingly.
- A famous example is the itemset "diapers and beer".
  - *Strategy:* run a sale on diapers and simultaneously raise the price of beer.
  - *Strategy:* place cans of beer in the aisle for baby products.

Plagiarism Detection

Items: Documents.

Baskets: Sentences.

Clinical Analysis

Items Drugs and side-effects.

Baskets: Patients.

- Has been used to detect combinations of drugs that result in particular side-effects.
- This approach requires an extension: Absence of an item needs to be observed as well as presence.

1.3 Finding Association Rules

Measures of Performance

Confidence
The % of antecedent transactions that also have the consequent item set.

- Confidence is the conditional probability of taking the consequent given that we have taken the antecedent:
$$\begin{aligned} Confidence &= P(consequent|antecedent) \\ &= \tfrac{P(antecedent\&consequent)}{P(antecedent)} \end{aligned}$$
- However, if the antecedent or consequent have strong support, this value is not useful. This is because if the probability of either the antecedent or the consequent is very high, the conditional probability would be high even if the antecedent and the consequent are independent of each other.

Lift Ratio

$$Lift\ ratio = \frac{Confidence}{Benchmark\ confidence}$$

- The benchmark confidence is the percentage of transactions in the entire dataset that have the consequent:
$$Benchmark\ confidence = \frac{no.\ transactions\ with\ consequent\ item\ set}{no.\ transactions\ in\ database}$$
- Lift ratio > 1 indicates that the rule is more useful than just selecting transactions randomly.

Process of Rule Selection

Generate all rules that meet specified support & confidence.

1. Find frequent item sets (those with sufficient support — see above).
2. From these item sets, generate rules with sufficient confidence (or lift ratio, whatever criteria we set).

   Example

   Rules from {red, white} item set.

   - Support for {red,white} = 40%; Support for {red} = 50%; Support for {white} = 80%.

   The rules are:

   (a) IF {red} THEN {white}
      - Confidence = 40% / 50% = 80%.
      - Lift ratio = 80% / 80% = 1.

   (b) IF {white} THEN {red}
      - Support for {red,white} = 40%.
      - Confidence = 40% / 80% = 50%.
      - Lift ratio = 50% / 50% = 1.

   If confidence cutoff is 70%, report only rule 1. If lift ratio > 1, none of these rules are useful.

Interpretation

- *Lift ratio* shows how effective the rule is in finding consequents (useful if finding particular consequents is important).
- *Confidence* shows the rate at which consequents will be found (useful in learning costs of promotion).
- *Support* measures overall impact.

Caution: The Role of Chance

- Random data can generate apparently interesting association rules.
- The more rules you produce, the greater this danger.
- Rules based on large numbers of records are less subject to this danger.

2 Finding Frequent Itemsets

2.1 The A-Priori Algorithm

Finding Frequent Itemsets

- A naïve approach to find frequent itemsets of length k would use a nested loop that generates all possible itemsets.
- But in a real scenario there may be many possible different items.
- This approach would be too time-consuming and may exceed memory capacity.
  - Suppose 10^5 items, counts are 4-byte integers.
  - Number of pairs of items: $10^5(10^5 - 1)/2 = 5 * 10^9$.
  - This would require $2 * 10^{10}$ bytes (20Gb of memory).
  - *What about counting triples??*

Intuition of the A-Priori Algorithm

Monotonicity

- If a set of items I appears at least s times, *so does every subset J of I*.
- Consequently: if a set of items J has a support below s, *so does every set I containing J*.
- Applied to a pair of items: If an item i does not appear in s baskets, then no pair including i can appear in s baskets.

Generating Frequent Item Sets

The A-Priori Algorithm
For k products …

1. User sets a minimum support criterion.
2. *First pass*: generate list of one-item sets that meet the support criterion.
3. *Second pass:* use the list of one-item sets to generate list of two-item sets that meet the support criterion.
4. *Third pass:* use list of two-item sets to generate list of three-item sets.
5. …
6. Continue up through k-item sets.

Example with the Phone Faceplates Dataset

The Dataset

Transaction	Faceplate colours purchased			
1	red	white	green	
2	white	orange		
3	white	blue		
4	red	white	orange	
5	red	blue		
6	white	blue		
7	white	orange		
8	red	white	blue	green
9	red	white	blue	
10	yellow			

One-item sets with support ≥ 3

Set	Support
red	*5*
white	*8*
green	2
orange	*3*
blue	*5*
yellow	1

Two-item sets ≥ 3

Set	Support
red, white	*4*
red, orange	1
red, blue	*3*
white, orange	*3*
white, blue	*4*
orange, blue	0

Three-item sets

Set	Support
red, white, blue	2

2.2 An Efficient Implementation of A-Priori

Considerations for Efficiency

- We assume that there are too many baskets to fit in main memory.
- So the file of baskets must be read from disk.
- We also assume that the number of items in each basket is relatively small.
- It will generally take much longer time to read a basket than to compute pairs or triples of items in the basket.
- There will normally be very few itemsets with size larger than 3 and support over the threshold.

We'd better read the basket file sequentially, and as few times as possible.

A-Priori Algorithm — Pass 1

```
1. Initialise counters for all items to zero.
2. FOR each basket:
3.     Read basket from disk.
4.     FOR each item in basket:
5.         Increment item's count.
```

Pass 1 — Example

The following three slides illustrate the progress of pass 1 after processing the first basket, after processing the second basket, and after processing all baskets.

Pass 1 — Example

The Dataset

Transaction	Faceplate colours purchased			
1	red	white	green	
2	white	orange		
3	white	blue		
4	red	white	orange	
5	red	blue		
6	white	blue		
7	white	orange		
8	red	white	blue	green
9	red	white	blue	
10	yellow			

One-item sets with support ≥ 3

Set	Support
red	1
white	1
green	1
orange	0
blue	0
yellow	0

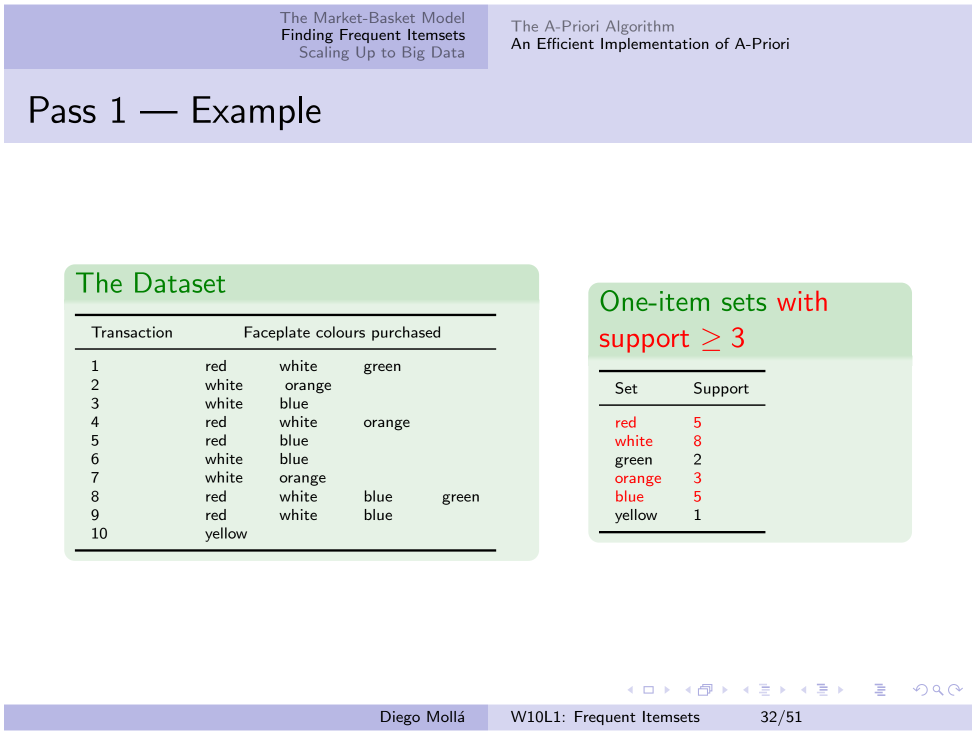
Pass 1 — Example

The Dataset

Transaction	Faceplate colours purchased			
1	red	white	green	
2	white	orange		
3	white	blue		
4	red	white	orange	
5	red	blue		
6	white	blue		
7	white	orange		
8	red	white	blue	green
9	red	white	blue	
10	yellow			

One-item sets with support ≥ 3

Set	Support
red	1
white	2
green	1
orange	1
blue	0
yellow	0



Pass 1 — Example

The Dataset

Transaction	Faceplate colours purchased			
1	red	white	green	
2	white	orange		
3	white	blue		
4	red	white	orange	
5	red	blue		
6	white	blue		
7	white	orange		
8	red	white	blue	green
9	red	white	blue	
10	yellow			

One-item sets with support ≥ 3

Set	Support
red	5
white	8
green	2
orange	3
blue	5
yellow	1

A-Priori Algorithm — Storage Considerations for Pass 2

- Storing all possible pairs of items might not fit on memory.
- So, we only store pairs of items that we are counting.
- Pairs of items are stored as triples:

$$(i, j, c)$$

i: Item 1

j: Item 2

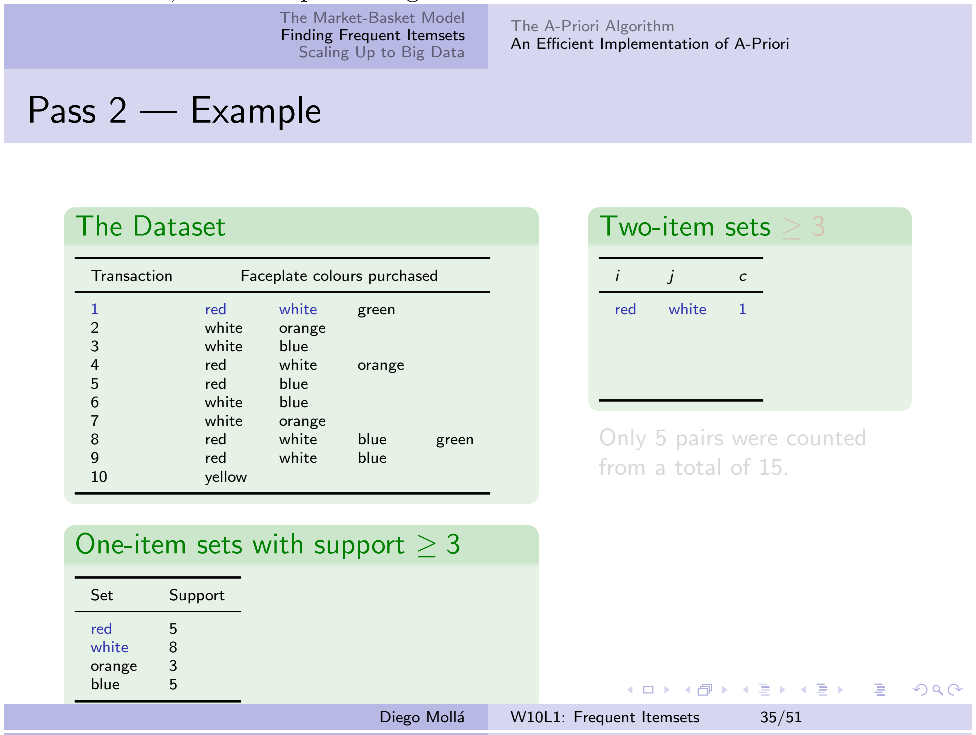
c: Count (initialised to zero)

A-Priori Algorithm — Pass 2

```
1. FOR each basket:
2.     Read basket from disk.
3.     FOR each pair $(i, j)$ of items in basket:
4.         IF both $i$ and $j$ are frequent (according to first pass):
5.         IF $(i, j)$ has been counted:
6.             Increment count for $(i, j)$
7.         ELSE:
8.             Add counter for $(i, j)$ initialised to 1.
```

Pass 2 — Example

The following three slides illustrate the progress of pass 1 after processing the first basket, after processing the second basket, and after processing all baskets.



Pass 2 — Example

The Dataset

Transaction	Faceplate colours purchased			
1	red	white	green	
2	white	orange		
3	white	blue		
4	red	white	orange	
5	red	blue		
6	white	blue		
7	white	orange		
8	red	white	blue	green
9	red	white	blue	
10	yellow			

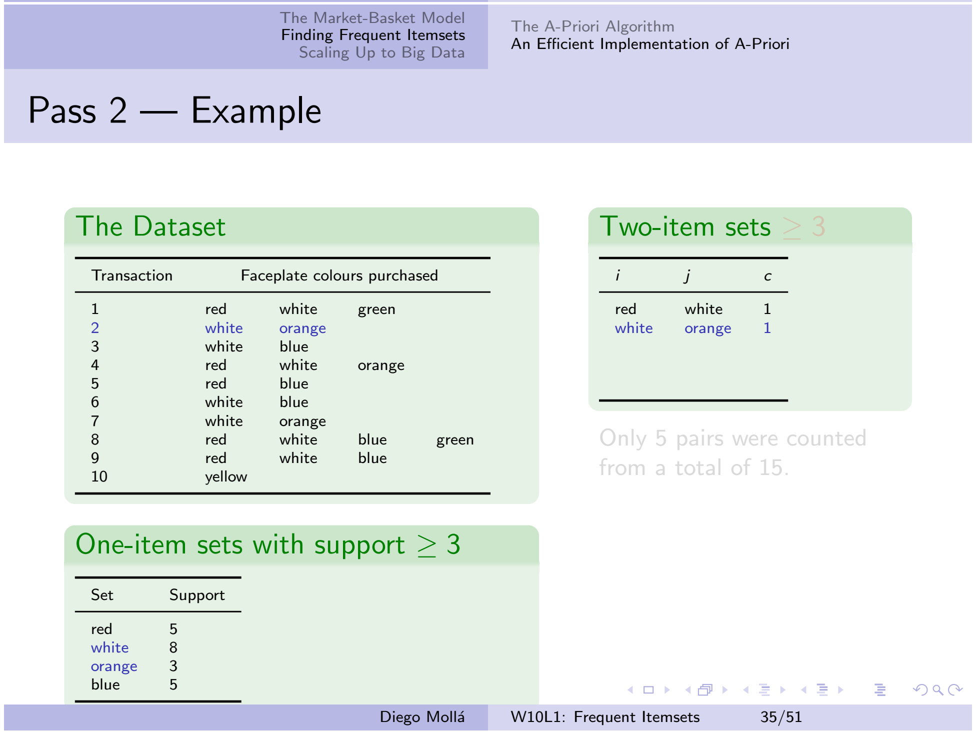
One-item sets with support ≥ 3

Set	Support
red	5
white	8
orange	3
blue	5

Two-item sets ≥ 3

i	j	c
red	white	1

Only 5 pairs were counted from a total of 15.



Pass 2 — Example

The Dataset

Transaction	Faceplate colours purchased			
1	red	white	green	
2	white	orange		
3	white	blue		
4	red	white	orange	
5	red	blue		
6	white	blue		
7	white	orange		
8	red	white	blue	green
9	red	white	blue	
10	yellow			

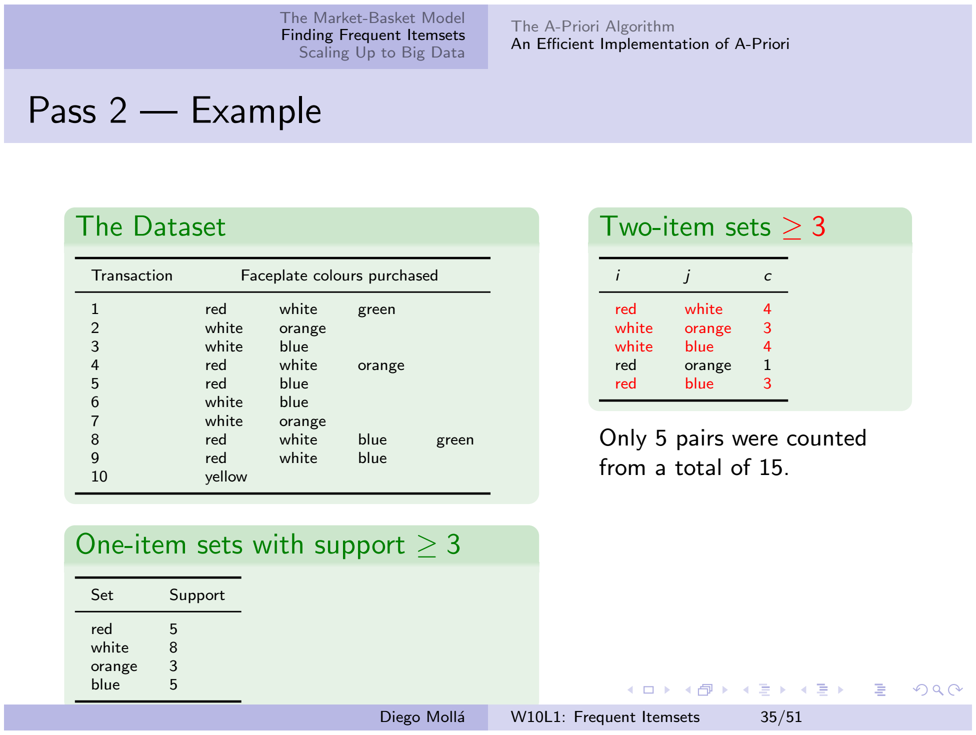
One-item sets with support ≥ 3

Set	Support
red	5
white	8
orange	3
blue	5

Two-item sets ≥ 3

i	j	c
red	white	1
white	orange	1

Only 5 pairs were counted from a total of 15.



Pass 2 — Example

The Dataset

Transaction	Faceplate colours purchased			
1	red	white	green	
2	white	orange		
3	white	blue		
4	red	white	orange	
5	red	blue		
6	white	blue		
7	white	orange		
8	red	white	blue	green
9	red	white	blue	
10	yellow			

One-item sets with support ≥ 3

Set	Support
red	5
white	8
orange	3
blue	5

Two-item sets ≥ 3

i	j	c
red	white	4
white	orange	3
white	blue	4
red	orange	1
red	blue	3

Only 5 pairs were counted from a total of 15.

3 Scaling Up to Big Data

3.1 PCY Algorithm

Park-Chen-Yu Algorithm

Observation of A-Priori algorithm

- In pass 1, most memory is idle.
- Pass 2 will usually be the most memory-consuming.

⇒ *Can we use the idle memory to reduce memory required in pass 2?*

Solution by PCY
At pass 1, while counting items, maintain a hash table that will count pairs of items.

PCY Algorithm — First Pass

Algorithm

```
1. FOR each basket:
2.    FOR each item in basket:
3.       Add 1 to item's count.
4.    FOR each pair of items in basket:
5.       Hash the pair to a bucket.
```

```
6.        *Add 1 to the count for that bucket.*
```

Notes

1. We want to use as many hash buckets as we can fit in memory.
2. Generalisation of a Bloom filter: We need to count the pairs so that we can check if their support is at least s.

PCY Algorithm — Second Pass

Algorithm

```
1. FOR each basket:
2.    FOR each pair in the basket:
3.       IF both items in the pair are frequent:
4.          Use hash function of pass 1 to hash the pair to a bucket.
5.          IF the count for that bucket is $\geq s$:
6.             Count the pair.
```

Note

- Several pairs might have hashed to the same bucket.

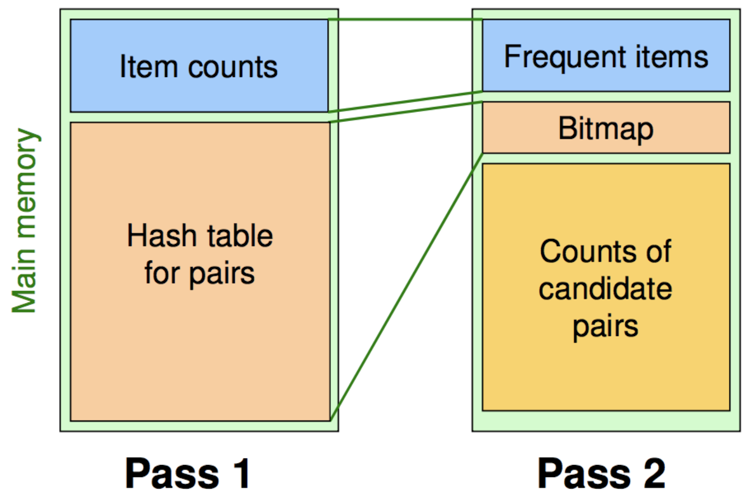
⇒ Even if a pair hashes to a bucket with count $\geq s$, it might not be frequent.

⇒ If a pair does not hash to a bucket with count $\geq s$, we are certain that it is not frequent.

What is the Advantage of Hash Tables?

- Like in the Bloom filter, the space to store the buckets in memory is smaller than the space of storing pairs of items and their counts.
- This is so because the number of buckets is less than the number of pairs of items.
- We only need to know what buckets are over a threshold, so after the first pass we can compact the hash table into a bit vector.
  - Bit = 1 if bucket is $\geq s$.
- We can actually chain hash tables to reduce the number of pairs to count... see below.

Main Memory: Picture of PCY



http://www.mmds.org

3.2 Multistage Algorithm

The Multistage Algorithm

- The multistage algorithm attempts to reduce the number of pairs to be counted.
- It does this by adding a new pass through the file of baskets.

Pass 1
Count individual items and maintain a hash table of counts of pairs as in PCY algorithm.

Pass 2
Create a new hash table, using an independent hash function, that counts pairs whose items were hashed as frequent in pass 1 *and both items in the pair are frequent as determined by pass 1.*

Pass 3
Count pairs that hashed to frequent buckets *in both hashes.*

Multistage Algorithm — Pass 1

Algorithm

```
1. FOR each basket:
2.     FOR each item in basket:
3.         Add 1 to item's count.
4.     FOR each pair of items in basket:
5.         Use hash function 1 to hash the pair to a bucket.
6.         Add 1 to the count for that bucket.
```

Note
This is exactly the same as pass 1 of PCY.

Multistage Algorithm — Pass 2

Algorithm

```
1. FOR each basket:
2.     FOR each pair in the basket:
3.         IF both items in the pair are frequent:
4.             Use hash function 1 to hash the pair to a bucket.
5.             IF the count for that bucket is $\geq s$:
6.                 Use hash function 2 to hash the pair to a bucket.
7.                 Add 1 to the count for that bucket..
```

Note
We keep two separate hash tables, one per hash function.

Multistage Algorithm — Pass 3

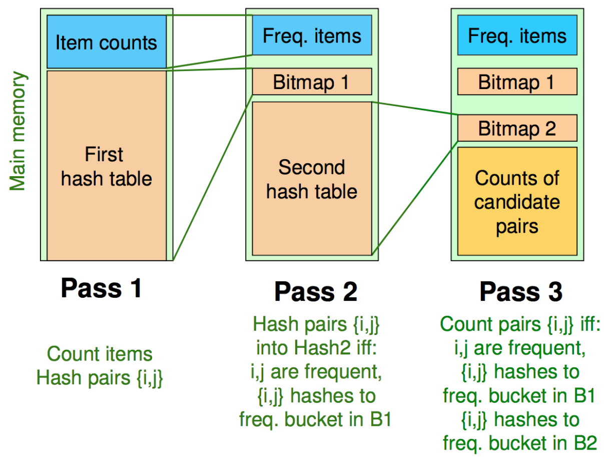
Algorithm

```
1. FOR each basket:
2.     FOR each pair in the basket:
3.         IF both items in the pair are frequent:
4.             Use hash function 1 to hash the pair to a bucket.
5.             Use hash function 2 to hash the pair to a bucket.
6.             IF the count for both buckets is $\geq s$:
7.                 Count the pair.
```

Do we really need to maintain both hashes in pass 3?

- It would be tempting to avoid checking the first hash in pass 3.
- But there may be pairs that hash to a frequent bucket in hash 2 but do not hash to a frequent bucket in pass 1.
- Remember: due to collisions, an infrequent pair may end up hashed in the same bucket as a frequent pair.
- We therefore need to make sure that the two hash functions are independent.

Main Memory: Multistage


Main memory
Item counts
First hash table
Freq. items
Bitmap 1
Second hash table
Freq. items
Bitmap 1
Bitmap 2
Counts of candidate pairs
Pass 1
Count items
Hash pairs {i,j}
Pass 2
Hash pairs {i,j} into Hash2 iff: i,j are frequent, {i,j} hashes to freq. bucket in B1
Pass 3
Count pairs {i,j} iff: i,j are frequent, {i,j} hashes to freq. bucket in B1 {i,j} hashes to freq. bucket in B2

http://www.mmds.org

Take-home Messages

- The Marked-Basket model.
- General approach to find association rules.
- *Counting item pairs are the costliest part of finding association rules.*
- The A-Priori algorithm.
- Efficient implementation of A-Priori.
- Variants for big data.

What's Next

Week 11

- Large-Scale Machine Learning (I)