Lecture 11
Frequent Itemsets II

Alexander Schönhuth

UNIVERSITÄT BIELEFELD
Faculty of Technology

Bielefeld University
June 1, 2023

TODAY

Overview: Frequent Itemsets II

- Mining Frequent Itemsets: Recap
- The Algorithm of Park, Chen and Yu (PCY)
- The Multistage Algorithm
- The Multihash Algorithm
- Toivonen's Algorithm

Learning Goals: Understand these topics and get familiarized

Mining Frequent Itemsets
Recap

Frequent Itemsets: Overview

Foundations

- There are *items* available in the market
- There are *baskets*, sets of items having been purchased together
- A *frequent itemset* is a set of items that is found to commonly appear in many baskets
- The *frequent-itemset problem* is to identify frequent itemsets

Frequent Itemsets: Definition

Definition [Frequent Itemset]:

- Let $s > 0$ be a *support threshold*
- Let I be a set of items
- $\text{supp}(I)$, the *support* of I, is the number of baskets in which I appears as a subset

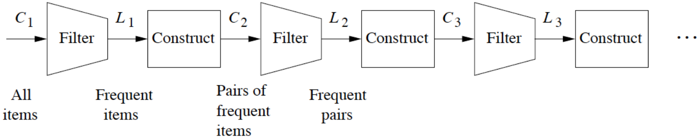
An itemset I is referred to as *frequent* if

$$\text{supp}(I) \geq s \tag{1}$$

that is, if the support of I is at least the support threshold

A Priori Algorithm
Recap

A-PRIORI ALGORITHM: CANDIDATE GENERATION AND FILTERING

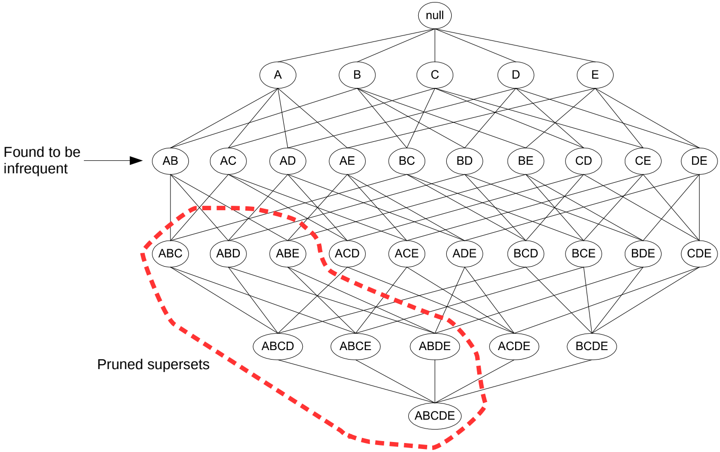


A-Priori algorithm: Alternating between candidate generation and filtering

Adopted from `mmds.org`

- *Construct:* C_k itemsets of size k, whose $k-1$-subsets belong to L_{k-1}
  - E.g. C_2 pairs of single items that are members of L_1, so are frequent
- *Filter:* Members of C_k whose count exceeds s belongs to L_k
- *Bottleneck:* Size of C_2, the candidate pairs
  - Why? Monotonicity implies in practice(!) that $|C_2| > |C_3| > |C_4| > ...$
  - Although $\binom{n}{2}$ (possible pairs) $< \binom{n}{3}$ (possible triples) $< \binom{n}{4}$...

Monotonicity to the Rescue

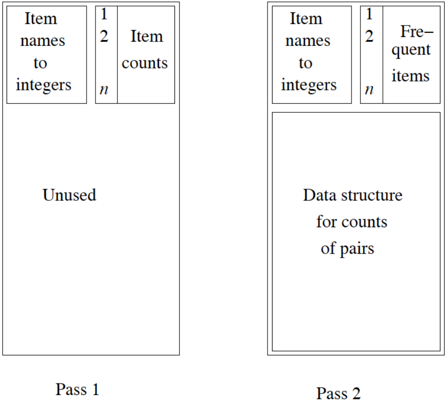


Itemsets for items A,B,C,D,E

Neglecting supersets of infrequent pair {A,B}

Adopted from `mmds.org`

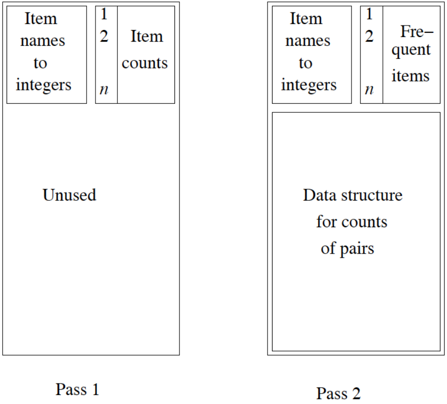
A-Priori Generating C_2: Main Memory Usage



Use of main memory during A-Priori passes
Pass 1: Generating L_1; Pass 2: Generating L_2

Adopted from `mmds.org`

A-Priori Generating C_2: Main Memory Usage



Adopted from `mmds.org`

From C_3 main memory no longer an issue

☞ naive approaches work, one pass enough

A-Priori Algorithm Extensions
The PCY Algorithm

Bottleneck: Size of C_2

- The predominant bottleneck in most applications of A-Priori is the size of C_2, the candidate pairs
- Several algorithms address to trim down that size
- Exemplary algorithms:
  - The algorithm of Park, Chen and Yu *(PCY algorithm)*
  - The Multistage algorithm
  - The Multihash algorithm
- We will treat all algorithms in the following

THE PCY ALGORITHM

- *Observation:* Much of main memory during first pass of A-Priori remains unused
- Use that space for a hash table H that
  - hashes pairs of items $\{i, j\}$ to
  - buckets holding integers $H[\{i, j\}] \in \mathbb{N}$, where

$$H[\{i, j\}] \quad \text{is number of times any pair hashed to that bucket} \qquad (2)$$

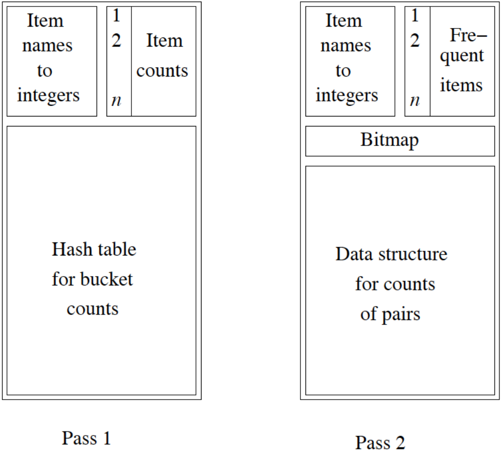
- To construct H, use double loop through baskets:
  - hash each resulting pair to bucket
  - increase the integer in that bucket by one
- A *frequent bucket b* exceeds the support threshold s

The PCY Algorithm

- A *frequent bucket b* exceeds the support threshold s
- So, for any bucket b:
  - If b is infrequent, none of the pairs that hashed to b are frequent
  - If b is frequent, pairs hashing to it could be frequent
- *Definition of candidate set C_2:* For $\{i, j\} \in C_2$, both
  - i and j must be frequent
  - $\{i, j\}$ must hash to a frequent bucket
- *Use of H in second pass:*
  - Transform H into bitmap H'

$$H'[\{i,j\}] = \begin{cases} 1 & \text{if } H[\{i,j\}] \geq s \\ 0 & \text{if } H[\{i,j\}] < s \end{cases} \quad (3)$$

PCY Algorithm: Main Memory Usage



Use of main memory during A-Priori passes

Adopted from `mmds.org`

The Multistage Algorithm

The Multistage Algorithm

- *Particular Motivation:* Selecting $\{i, j\}$ to be in C_2
- In PCY: even when reducing to frequent i and j, and $\{i, j\}$ hashing to frequent buckets, still too many pairs to be counted
- So, need to decrease size of C_2 further
- Do this by introducing extra pass:
  - *First pass:* as before in PCY
  - *Second pass:* create another hash table raising a third condition
  - *Third pass:* count only pairs that fulfill all three conditions

The Multistage Algorithm: Second Pass

- Second pass data structures from PCY:
  - List A on item names to integers
  - List C on frequent items: $C[i] = k$ if item i is k-th frequent item, and $C[i] = 0$ if i-th item is not frequent
  - Bitmap H': $H'[\{i,j\}] = 1$ iff $\{i,j\}$ hashed to frequent bucket
- Multistage second pass: consider only $\{i,j\}$, where
  - (*) both i and j are frequent
  - (**) $H'[\{i,j\}] = 1$, that is $\{i,j\}$ hashes to frequent bucket
  - Create H_2 hashing such $\{i,j\}$ to buckets holding integers

$$H_2[\{i,j\}] \in \mathbb{N}$$

The Multistage Algorithm: Second Pass

- To construct H_2, use double loop through baskets:
  - hash each pair that meets (*) and (**) to bucket, and
  - increase the integer in that bucket by one
- Again, a *frequent bucket b* in H_2 exceeds the support threshold s
- Relative to number of frequent buckets using first H, the number of frequent buckets in H_2 should be much reduced, because much less pairs are hashed

The Multistage Algorithm

- *Definition of Multistage C_2:* For $\{i, j\} \in C_2$, both
  - (*) i and j must be frequent
  - (**) $\{i, j\}$ must hash to a frequent bucket according to H
  - (***) $\{i, j\}$ must hash to a frequent bucket according to H_2
- *Use of C_2 in third pass:*
  - Keep A (items to integers), C (frequent items), H' (bitmap for H)
  - Transform H_2 into bitmap H'' where

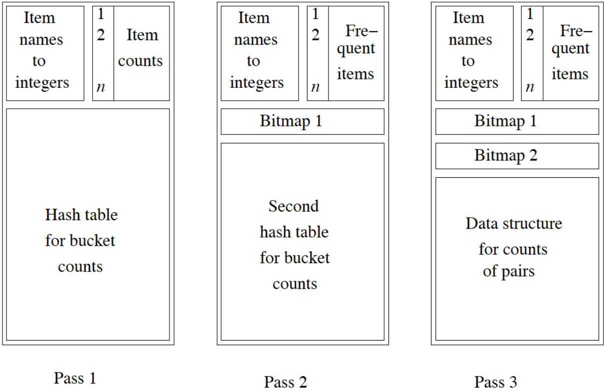
$$H''[b] = \begin{cases} 1 & \text{if } H_2[\{i, j\}] \geq s \\ 0 & \text{if } H_2[\{i, j\}] < s \end{cases} \tag{4}$$

where b is the bucket $\{i, j\}$ hashes to by H_2

THE MULTISTAGE ALGORITHM

- *(Tricky?) Question:* Why does (***) not imply (**) and (*)? Weren't all $\{i, j\}$ hashed with H_2 selected to hash to frequent bucket with H and consist of frequent i and j?
- *Answer:*
  - *Yes:* for the second part.
  - *But:* Any $\{i, j\}$ that does not consist of frequent i, j, or hash to frequent bucket with H could hash to frequent bucket with H_2 nevertheless, although not having contributed to count in the bucket it hashes to

MULTISTAGE ALGORITHM: MAIN MEMORY USAGE



Use of main memory during Multistage passes

Adopted from `mmds.org`

The Multihash Algorithm

The Multihash Algorithm

- *Particular Motivation:* Try to profit from virtues of Multistage algorithm in one, and not two passes
- So, in *first pass*, use two hash tables H_1 and H_2,
- Both H_1 and H_2 have only half as many buckets
- For proceeding with second pass, turn H_1 and H_2 into bitmaps H', H'' as in Multistage
- Apply exact same conditions as in Multistage for pair $\{i, j\}$ to be counted

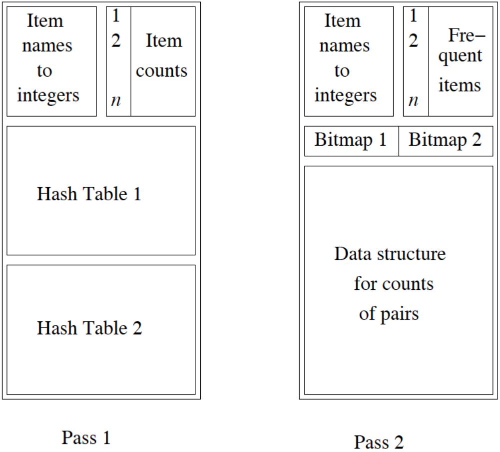
The Multihash Algorithm

- Both H_1 and H_2 have only half as many buckets
- That is like merging original buckets
- *Applicability:*
  - Majority of buckets infrequent
  - Average bucket size in PCY much lower than threshold s
  - ☞ Number of frequent buckets limited even when using half as many buckets

The Multihash Algorithm: Example

- Imagine average bucket count in PCY is $s/10$
- *Particular Assumption:* Number of pairs of items randomly hashing to frequent bucket is $1/10$
- So, with half as many buckets, average count in Multihash is $s/5$
- Number of pairs of items randomly hashing to frequent buckets with both H_1 and H_2 is $1/25$
- So, we deal with approximately 2.5 times less frequent pairs in Multihash than in PCY

Multihash Algorithm: Main Memory Usage



Use of main memory during Multihash passes

Adopted from `mmds.org`

Limited-Pass Algorithms

LIMITED-PASS ALGORITHMS

Strategy

- To save on main memory, consider only a subsample of baskets
- Take into account that one may have
  - *False negatives:* itemsets not identified as frequent although they are
  - *False positives:* itemsets identified as frequent although they are not
- In many applications, a certain amount of false negatives and/or positives is acceptable

Algorithms

- *Simple Randomized Algorithm:* basic strategy is briefly discussed
- *Savasere, Omiecinski, Navate (SON):* not considered in the following
- *Toivonen:* explained here

Simple Randomized Algorithm

Simple Randomized Algorithm: Strategy

- Let m be the overall number of baskets
- *Situation:* main memory can deal with only k baskets
- Select probability p such that $pm = k$
- Run through basket file, and select each basket to be part of sample with probability p
- If s is original support threshold, set $s' := sp$ for sample
- Run any A-Priori type algorithm on resulting subset of baskets using s' as support threshold
- Declare itemsets frequent in subsample as frequent overall

Simple Randomized Algorithm: Errors

- *False positive:* Itemset frequent in sample, but not in whole
- *False negative:* Itemset frequent in whole, but not in sample
- *Eliminating false positives:* Evaluate each itemset found to be frequent in sample by running through whole dataset
- *Eliminating false negatives:* Cannot eliminate false negatives entirely, but reduce them by choosing $s' < sp$, e.g. $s' = 0.9sp$

Toivonen's Algorithm

Toivonen's Algorithm I

Algorithm

- Run simple sample strategy at $s' = 0.9ps$ or $s' = 0.8ps$
- Construct all frequent itemsets from sampled baskets for support threshold s'
- Subsequently, construct *negative border* of itemsets in sample

Definition [Negative Border]:
An itemset I is in the *negative border* iff

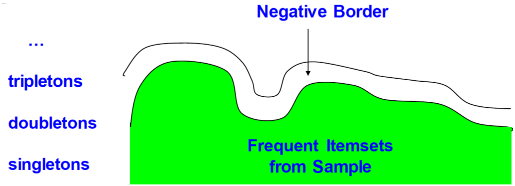
(i) I is not frequent, so $\text{supp}(I) < s'$

(ii) All $I' \subset I$ with $|I'| = |I| - 1$ are frequent, so $\text{supp}(I') \geq s'$

Negative Border

Definition [Negative Border]:
An itemset I is in the *negative border* iff

- I is not frequent, so $\text{supp}(I) < s'$
- All $I' \subset I$ with $|I'| = |I| - 1$ are frequent, so $\text{supp}(I') \geq s'$



Negative Border: Illustration

From `https://who.rocq.inria.fr/Vassilis.Christophides/Big/index.htm`

Negative Border: Example

- Consider items $\{A, B, C, D, E\}$
- Itemsets found to be frequent: $\{A\}, \{B\}, \{C\}, \{D\}, \{B, C\}, \{C, D\}$
- For formal reasons also the empty set $\emptyset$ is frequent
- Negative border:
  - $\{E\}$ not frequent, but $\emptyset$ is frequent ☞ $|\emptyset| = |\{E\}| - 1$ and $\emptyset$ only subset of $\{E\}$ qualifying for (ii) from definition two slides before
  - $\{A, B\}, \{A, C\}, \{A, D\}, \{B, D\}$: not frequent, but singletons contained in them, $\{A\}, \{B\}, \{C\}, \{D\}$, are
  - No triples in negative border (e.g. $\{B, D\}$ in $\{B, C, D\}$ not frequent)

Toivonen's Algorithm II

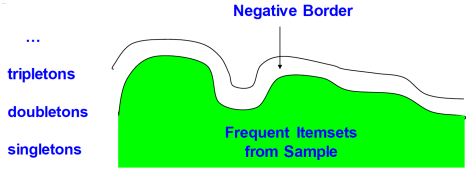
- *Pass through full dataset:* Count all itemsets, found to be frequent or in the negative border in the sample, in the whole
- Two possible outcomes:
  1. No member of negative border is frequent in whole dataset: frequent itemsets are frequent in sample and in whole
  2. Some member of negative border is frequent in whole dataset: there could be even larger sets frequent in the whole
     ☞ no guarantees, repeat the algorithm

Toivonen's Algorithm: Proof

- *Eliminating false positives:* As usual for simple randomized algorithms, by raising counts in the whole dataset, one can filter out itemsets that are frequent in the sample, but not in the whole dataset ✓
- *No false negatives:* If no member of the negative border is frequent in the whole dataset, show there is no itemset that
  - is frequent in the whole
  - while, in the sample not among the frequent itemsets

Toivonen's Algorithm: Proof

- *Proof of no false negatives:* Suppose the contrary: there is S
  - that is frequent in the whole
  - but not frequent in the sample
- By monotonicity, all subsets of S are frequent in the whole
- Choose $T \subseteq S$ of the *smallest* possible size such that still T is not frequent in the sample



Negative Border: Illustration

From https://who.rocq.inria.fr/Vassilis.Christophides/Big/index.htm

Toivonen's Algorithm: Proof

- *Claim:* T is in the negative border of the sample
- *Proof of Claim:*
  - All proper subsets of T are frequent in the sample, because T was chosen of the smallest possible size
  - T itself is not frequent in the sample
- We obtain that T was in the negative border of the sample, but frequent in the whole, which is a contradiction!

Materials / Outlook

- See *Mining of Massive Datasets*, sections 6.1–6.4
- As usual, see `http://www.mmds.org/` in general for further resources
- Next lecture: 'Recommendation Systems"
  - See *Mining of Massive Datasets*, 9.1, 9.3, 9.4