Frequent Itemsets II

Alexander Schönhuth

UNIVERSITÄT BIELEFELD
Faculty of Technology

Bielefeld University
July 22, 2021

TODAY

Overview

- Extensions of the A Priori Algorithm
  - The Multihash Algorithm
  - The Multistage Algorithm
- Limited Pass Algorithms
  - Simple Randomized 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

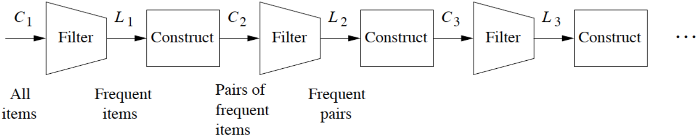
An itemset I is referred to as *frequent* if

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

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

A Priori Algorithm
Recap

A-PRIORI ALGORITHM: CANDIDATE GENERATION AND FILTERING

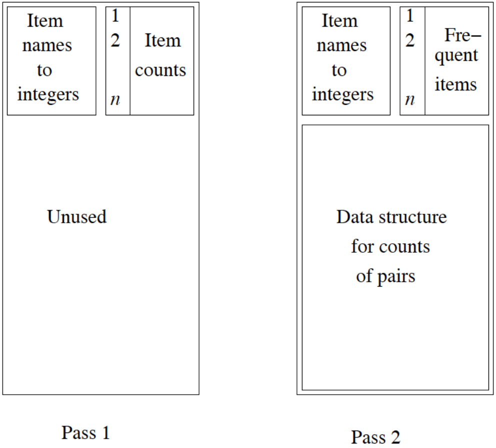


A-Priori algorithm: Alternating between candidate generation and filtering

Adopted from `mmds.org`

- *Construct:* Let C_k be all itemsets of size k, every $k-1$ of which belong to L_{k-1}
  - E.g. C_2 all pairs of items that are frequent themselves
- *Filter:* Make a *pass through baskets* to count members of C_k; those with count exceeding s will be part of L_k
- *Bottleneck:* Size of C_2, the candidate pairs

A-Priori Generating C_2: Main Memory Usage



Use of main memory during A-Priori passes

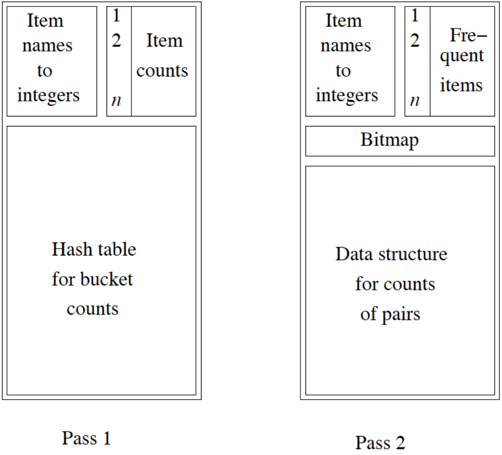
Adopted from `mmds.org`

A-Priori Algorithm Extensions

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
- Treated PCY algorithm last time
  - Additional criterion: *frequent buckets*
- Multistage and Multihash algorithm: today

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:
  - The *first pass* is as before in PCY
  - In the *second pass*, have another hash table that raises a third condition
  - In the *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 item pair $\{i, j\}$ mapped to frequent bucket
- Extra data structure Multistage second pass:
  - Hash table H_2 that hashing pairs of items $\{i, j\}$ to buckets holding integers

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

    if:
  - (*) both i and j are frequent
  - (**) $H'[\{i, j\}] = 1$, that is $\{i, j\}$ hashes to frequent bucket

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

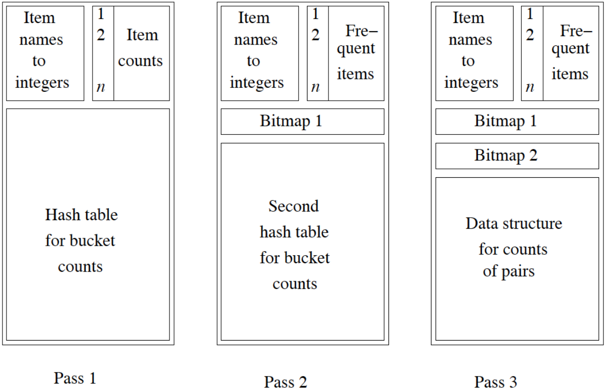
$$H''[b] = \begin{cases} 1 & \text{if } H_2[\{i, j\}] \geq s \\ 0 & \text{if } H_2[\{i, j\}] < s \end{cases} \tag{2}$$

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

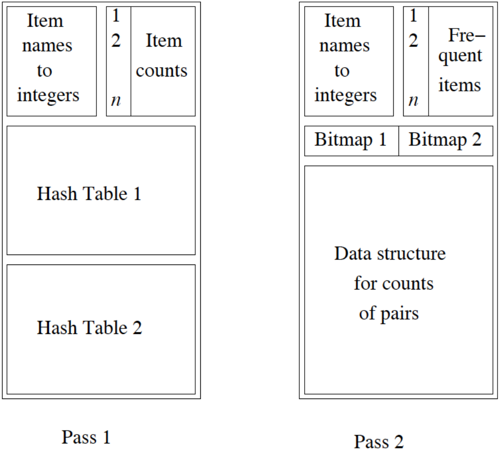
The Multihash Algorithm

- 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 $s/10$
- 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
- Consider a situation where 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 that is frequent in sample, but not in the whole
- *False negative:* Itemset that is frequent in the whole, but not in sample
- *Eliminating false positives:* Running through whole dataset and counting each itemset found to be frequent in the sample eliminates false positives entirely
- *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 (at support threshold s')
- Subsequently, construct *negative border* of itemsets in sample

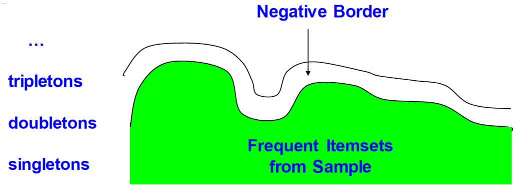
Definition [Negative Border]:
An itemset I is in the *negative border* iff

- 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

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
  - $\{A, B\}, \{A, C\}, \{A, D\}, \{B, D\}$: not frequent, but singletons contained are
  - No triples in negative border ($\{B, C, D\}$ is not, because $\{B, D\}$ is not frequent)

Toivonen's Algorithm II

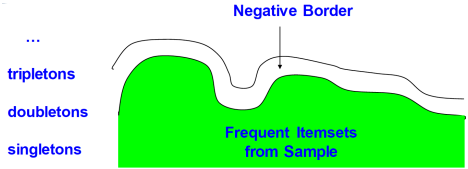
- *Pass through full dataset:* Count all itemsets found to be frequent or in the negative border in the sample
- Two possible outcomes:
    1. No member of negative border is frequent in whole dataset: correct set of itemsets frequent in the whole are the ones frequent in the sample found to be frequent in the 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

- *No false positives:* all frequent itemsets were determined as frequent in the whole dataset ✓
- *No false negatives:* If no member of the negative border is frequent in the whole dataset, we need to show that there is no itemset that
  - is frequent in the whole
  - while, in the sample not among the frequent itemsets
  - while, in the sample, not in the negative border

Toivonen's Algorithm: Proof

- *Proof of no false negatives:* Suppose the contrary. There is S
  - frequent in the whole
  - not frequent in the sample
  - not in the negative border
- 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*, 6.3.2, 6.3.3, 6.4.1, 6.4.2, 6.4.5, 6.4.6
- As usual, see `http://www.mmds.org/` in general for further resources