Improvements to A-Priori

Park-Chen-Yu Algorithm
Multistage and Multihash
Single-Pass Approximate Algorithms

Mining of Massive Datasets
Leskovec, Rajaraman, and Ullman
Stanford University

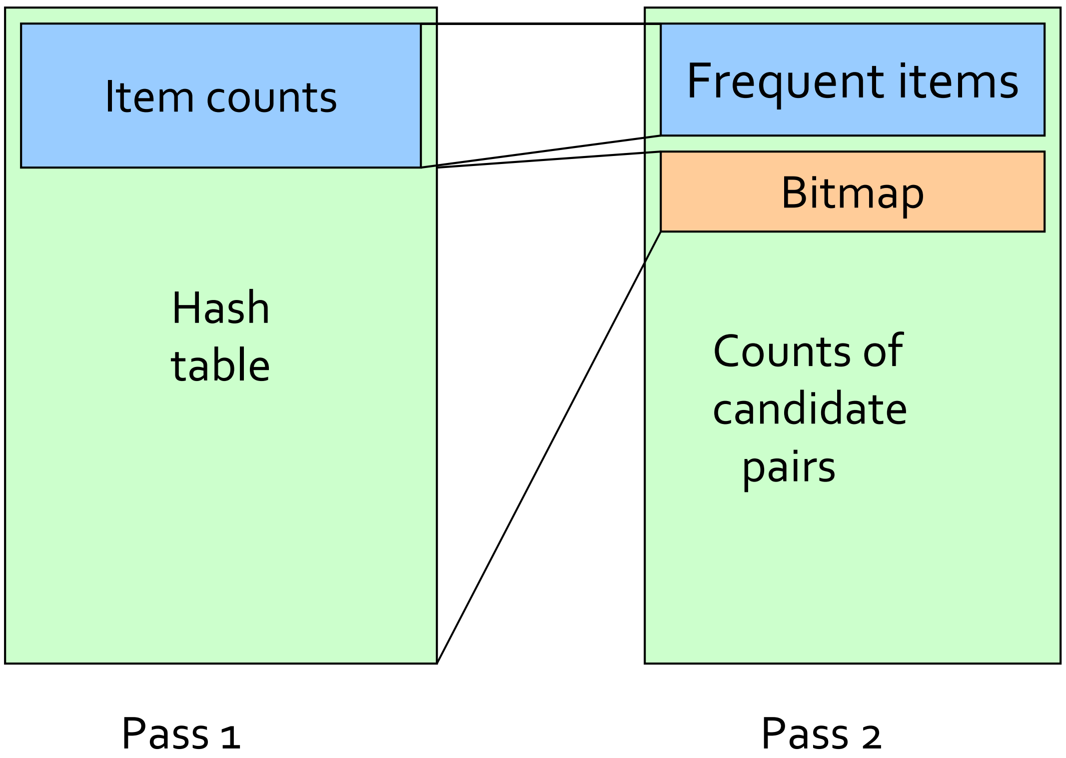
PCY Algorithm

- During Pass 1 of A-priori, most memory is idle.
- Use that memory to keep counts of buckets into which pairs of items are hashed.
  - Just the count, not the pairs themselves.
- For each basket, enumerate all its pairs, hash them, and increment the resulting bucket count by 1.

PCY Algorithm – (2)

- A bucket is *frequent* if its count is at least the support threshold.
- If a bucket is not frequent, no pair that hashes to that bucket could possibly be a frequent pair.
- On Pass 2, we only count pairs that hash to frequent buckets.

Picture of PCY



Pass 1: Memory Organization

- Space to count each item.
  - One (typically) 4-byte integer per item.
- Use the rest of the space for as many integers, representing buckets, as we can.

PCY 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
    }
}
```

Observations About Buckets

1. A bucket that a frequent pair hashes to is surely frequent.
   - We cannot use the hash table to eliminate any member of this bucket.
2. Even without any frequent pair, a bucket can be frequent.
   - Again, nothing in the bucket can be eliminated.

Observations – (2)

3. But in the best case, the count for a bucket is less than the support *s*.
   - Now, all pairs that hash to this bucket can be eliminated as candidates, even if the pair consists of two frequent items.

PCY Algorithm – Between Passes

- Replace the buckets by a bit-vector (the "bitmap"):
  - 1 means the bucket is frequent; 0 means it is not.
- 4-byte integers are replaced by bits, so the bit-vector requires 1/32 of memory.
- Also, decide which items are frequent and list them for the second pass.

PCY Algorithm – 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 number whose bit in the bit vector is 1.

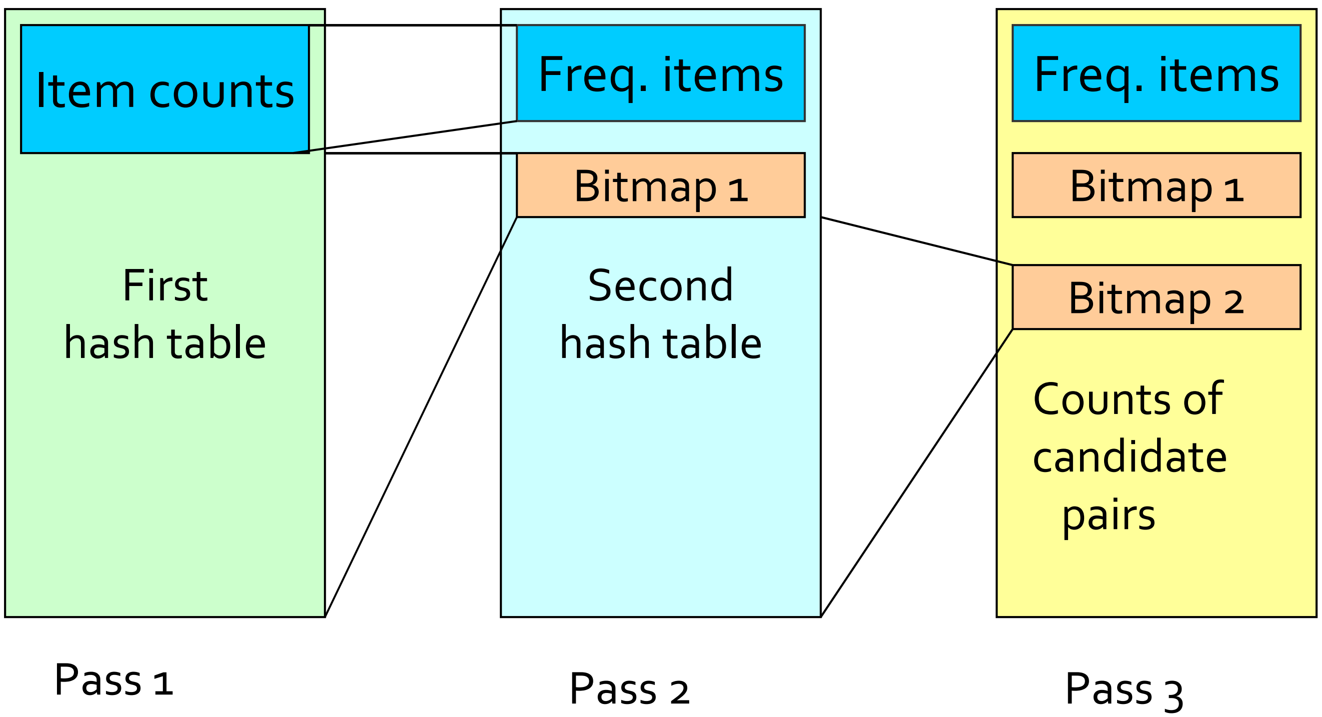
Memory Details

- Buckets require a few bytes each.
  - Note: we don't have to count past *s*.
  - # buckets is O(main-memory size).
- On second pass, a table of (item, item, count) triples is essential.
  - Thus, hash table must eliminate 2/3 of the candidate pairs for PCY to beat a-priori.

Multistage Algorithm

- Key idea: After Pass 1 of PCY, rehash only those pairs that qualify for Pass 2 of PCY.
- On middle pass, fewer pairs contribute to buckets, so fewer *false positives* – frequent buckets with no frequent pair.

Multistage Picture



Multistage – Pass 3

- Count only those pairs {*i*, *j*} that satisfy these candidate pair conditions:
  1. Both *i* and *j* are frequent items.
  2. Using the first hash function, the pair hashes to a bucket whose bit in the first bit-vector is 1.
  3. Using the second hash function, the pair hashes to a bucket whose bit in the second bit-vector is 1.

Important Points

1. The hash functions have to be independent.
2. We need to check both hashes on the third pass.
   - If not, we would wind up counting pairs of frequent items that hashed first to an infrequent bucket but happened to hash second to a frequent bucket.

Multihash

- Key idea: use several independent hash tables on the first pass.
- Risk: halving the number of buckets doubles the average count. We have to be sure most buckets will still not reach count *s*.
- If so, we can get a benefit like multistage, but in only 2 passes.

Multihash Picture