Roadmap

- Frequent Patterns
- A-Priori Algorithm
- Improvements to A-Priori
  - Park-Chen-Yu Algorithm
  - Multistage Algorithm
  - Approximate Algorithms
  - Compacting Results

PCY Algorithm

- Hash-based improvement to A-Priori.
- 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.
- Gives extra condition that candidate pairs must satisfy on Pass 2.

PCY Algorithm --- Before Pass 1 Organize Main Memory

- 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)
    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. If a bucket contains a frequent pair, then the bucket 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.

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:
  - 1 means the bucket count exceeds the support $s$ (frequent bucket); 0 means it did not.

- Integers are replaced by bits, so the bit-vector requires little second-pass space.

- 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:
  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.

- Notice all these conditions are necessary for the pair to have a chance of being frequent.
Memory Details

- Hash table requires buckets of 2-4 bytes.
  - Number of buckets thus almost 1/4-1/2 of the number of bytes of main memory.

- On second pass, a table of (item, item, count) triples is essential.
  - Thus, hash table must eliminate 2/3 of the candidate pairs 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

Count only those pairs \(\{i, j\}\) that satisfy:

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 two 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.
**Extensions**

- Either multistage or multihash can use more than two hash functions.

- In multistage, there is a point of diminishing returns, since the bit-vectors eventually consume all of main memory.

- For multihash, the bit-vectors total exactly what one PCY bitmap does, but too many hash functions makes all counts $\geq s$. 
All (Or Most) Frequent Itemsets In $< 2$ Passes

- Simple algorithm.
- SON (Savasere, Omiecinski, and Navathe).
- Toivonen.

Simple Algorithm --- (1)

- Take a main-memory-sized random sample of the market baskets.
- Run a-priori or one of its improvements (for sets of all sizes, not just pairs) in main memory, so you don’t pay for disk I/O each time you increase the size of itemsets.
  - Be sure you leave enough space for counts.
The Picture

Copy of sample baskets

Space for counts

Simple Algorithm --- (2)

- Use as your support threshold a suitable, scaled-back number.
  - E.g., if your sample is 1/100 of the baskets, use $s/100$ as your support threshold instead of $s$. 
Simple Algorithm --- Option

- Optionally, verify that your guesses are truly frequent in the entire data set by a second pass.

- But you don’t catch sets frequent in the whole but not in the sample.
  - Smaller threshold, e.g., $s/125$, helps.

SON Algorithm --- (1)

- Repeatedly read small subsets of the baskets into main memory and perform the first pass of the simple algorithm on each subset.

- An itemset becomes a candidate if it is found to be frequent in any one or more subsets of the baskets.

SON Algorithm --- (2)

- On a second pass, count all the candidate itemsets and determine which are frequent in the entire set.

- **Key “monotonicity” idea:** an itemset cannot be frequent in the entire set of baskets unless it is frequent in at least one subset.

Toivonen’s Algorithm --- (1)

- Start as in the simple algorithm, but lower the threshold slightly for the sample.
  - **Example:** if the sample is 1% of the baskets, use \( s/125 \) as the support threshold rather than \( s/100 \).
  - Goal is to avoid missing any itemset that is frequent in the full set of baskets.

- H. Toivonen. *Sampling large databases for association rules.* In *VLDB'96*
Toivonen’s Algorithm --- (2)

- Add to the itemsets that are frequent in the sample the *negative border* of these itemsets.

- An itemset is in the negative border if it is not deemed frequent in the sample, but *all* its immediate subsets are.

Example: Negative Border

- **$ABCD$** is in the negative border if and only if it is not frequent, but all of **$ABC$**, **$BCD$**, **$ACD$**, and **$ABD$** are.
Toivonen’s Algorithm --- (3)

- In a second pass, count all candidate frequent itemsets from the first pass, and also count the negative border.

- If no itemset from the negative border turns out to be frequent, then the candidates found to be frequent in the whole data are **exactly** the frequent itemsets.

Toivonen’s Algorithm --- (4)

- What if we find something in the negative border is actually frequent?

- We must start over again!

- Try to choose the support threshold so the probability of failure is low, while the number of itemsets checked on the second pass fits in main-memory.
Theorem:

- If there is an itemset frequent in the whole, but not frequent in the sample, then there is a member of the negative border frequent in the whole.

Proof:

- Suppose not; i.e., there is an itemset $S$ frequent in the whole, but not frequent or in the negative border in the sample.

- Let $T$ be a smallest subset of $S$ that is not frequent in the sample.

- $T$ is frequent in the whole (monotonicity).

- $T$ is in the negative border (else not “smallest”).
Compacting the Output

- A long pattern contains a combinatorial number of sub-patterns, e.g., \( \{a_1, ..., a_{100}\} \) contains \( \binom{100}{1} + \binom{100}{2} + ... + \binom{1}{0} = 2^{100} - 1 = 1.27 \times 10^{30} \) sub-patterns!

- Solution: *Mine closed patterns and max-patterns instead*

1. **Maximal Frequent itemsets**: no immediate superset is frequent.

2. **Closed itemsets**: no immediate superset has the same count.
   - Stores not only frequent information, but exact counts.

Closed Patterns and Max-Patterns

- An itemset \( X \) is **closed** if \( X \) is frequent and there exists no super-pattern \( Y \supset X \), with the same support as \( X \) (proposed by Pasquier, et al. @ ICDT’99)

- An itemset \( X \) is a **max-pattern** if \( X \) is frequent and there exists no frequent super-pattern \( Y \supset X \) (proposed by Bayardo @ SIGMOD’98)

- Closed pattern is a lossless compression of freq. patterns
  - Reducing the # of patterns and rules
Example: Maximal/Closed

<table>
<thead>
<tr>
<th>Count</th>
<th>Maximal s=3</th>
<th>Closed</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>B</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>C</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>AB</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>AC</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>BC</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ABC</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Closed Patterns and Max-Patterns

- Exercise. DB = \{<a_1, ..., a_{100}>, <a_1, ..., a_{50}>\}
  - Min_sup = 1.
- What is the set of closed itemsets?
Closed Patterns and Max-Patterns

- Exercise. DB = \{<a_1, ..., a_{100}>, <a_1, ..., a_{50}>\}
  - Min_sup = 1.
- What is the set of closed itemsets?
  - <a_1, ..., a_{100}>: 1
  - <a_1, ..., a_{50}>: 2

- What is the set of max-patterns?
Closed Patterns and Max-Patterns

- Exercise. DB = \{<a_1, \ldots, a_{100}>, <a_1, \ldots, a_{50}>\}
  - Min_sup = 1.
- What is the set of closed itemsets?
  - \(<a_1, \ldots, a_{100}>\): 1
  - \(<a_1, \ldots, a_{50}>\): 2
- What is the set of max-patterns?
  - \(<a_1, \ldots, a_{100}>\): 1

- Exercise. DB = \{<a_1, \ldots, a_{100}>, <a_1, \ldots, a_{50}>\}
  - Min_sup = 1.
- What is the set of closed itemsets?
  - \(<a_1, \ldots, a_{100}>\): 1
  - \(<a_1, \ldots, a_{50}>\): 2
- What is the set of max-patterns?
  - \(<a_1, \ldots, a_{100}>\): 1
- What is the set of all patterns?
Ref: Basic Concepts of Frequent Pattern Mining

- **Association Rules** R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. SIGMOD’93.
- **Max-pattern** R. J. Bayardo. Efficiently mining long patterns from databases. SIGMOD’98.
- **Sequential pattern** R. Agrawal and R. Srikant. Mining sequential patterns. ICDE’95

Ref: Apriori and Its Improvements