1 September 4th, 2019

1.1 NP-Completeness of Finding Large Itemsets

The problem of finding all "large" itemsets is equivelent to finding all "large" J-itemsets for each positive integer J. (for our case this is finding J-itemsets with support ≥ 3).

However, this problem is NP-Complete, as it is equivelent to solving the **Balanced Complete bipartite Subgraph**, a known NP-Complete problem.

Definition 1.1. The **Balanced Complete Bipartitie Subgraph** is as follows:

- Instance: Given a bipartite graph $G = (V, E)$ and positive integer $K \leq |V|$
- Question: Are there two disjoint subsets $V_1, V_2 \subseteq V$ such that $|V_1| = |V_2| = K$ and such that, for each $u \in V_1$ and each $v \in V_2$, $(u, v) \in E$.

Figure 1: $V_1 = \{A, B, C\}, V_2 = \{E, F, G\}$ for the left, no such V_1, V_2 exists for the right

The reduction from the graph problem to the itemset problem is as follows:

- For each vertex in V_1, create a transaction
- For each vertex in V_2, create a item
- For each edge (u, v), create a purchase of item v in transaction u
- We have $f = K$ and $J = K$

Remark 1.2 — In other words, solving the graph problem is equivelent tothe itemset problem, i.e. is there a K-frequent itemset of size K. Since this is a restriction of the itemset problem, if we can solve the itemset problem, we can solve the Balanced Complete Bipartite Subgraph problem. As such, the itemset problem is also NP-Complete, since BCBS is NP-Complete.

Remark 1.3 — NP-Complete means that there is no polynomial time algorithm to solve it (unless P=NP).

1.2 Algorithm Aprior

This algorithm starts with "large" 1-itemsets, and iteratively builds "large" itemsets with bigger sizes (1-itemsets→ 2-itemset→ ...). It can be described as follows:

- Start with $L_1 = \{\{A\}, \{B\}, \{C\}, \{D\}, \{E\}\}$, i.e. the large 1-itemsets.
- From L_1, generate candidates for large 2-itemsets, C_2.
- From C_2, check and keep the large 2-itemsets, L_2.
- Repeat until the C_n is empty, i.e. there are no large itemsets of size n.
- The set of all large itemsets is $\bigcup_{i<n} L_i$.

Remark 1.4 — The reason why we want to do this is because reading the whole table frequently is slow, and this way, we can keep less things in memory.

To generate candidates for "large" n-itemsets from L_{n-1}, we can take note of the following properties:

Theorem 1.5
If an itemset S is large, then any proper subset of S must be large.

Proof. Note that the proper subsets of S are a relaxed version of S. Since we are relaxing the constraint, this property must be true. □

Theorem 1.6
If an itemset S is NOT large, then any proper superset of S must NOT be large.

Proof. Similarly, since the proper supersets are restricted versions of S, if S is not large, then any proper supersets of S must not be large, since it's even more restricted. □

Example 1.7
$\{B, C, E\}$ is large, thus $\{B, C\}, \{C, E\}\{C\}$ are all large (not exhaustive).

Using these properties, we can split the generation step into two steps:

Join : Suppose we know that the itemset $\{B, C\}$ and the itemset $\{B, E\}$ are large (i.e. in L_2), then $\{B, C, E\}$ is a potential candidate. In other words, from L_{n-1}, we join the $(n-1)$-itemsets to create the n-itemsets. Since our items can be ordered (as they are letters), we can check the **prefix** of the itemsets of all pairs of itemsets, and join together if they have the same prefix, where the prefix of the itemset is all items except the last.

> **Remark 1.8 —** The reason why we only need to check the prefix is because if we combine others, it will definitely be pruned.

Prune : Since having all proper subsets of S be large is a necessary condition for S to be large, we can prune the candidates generated from the Join steps by checking all proper subsets of one size smaller.

> **Remark 1.9 —** The reason why we only have to check the subsets that are one smaller is because the others would've been check at an earlier step. If they aren't large, then delete the candidate.

> **Remark 1.10 —** Note that this step can be done without reading the table, since we can just check if the subset s is in L_{k-1} and delete if not.

After we've generated the candidate large sets (C_i), we will check the database for the support of them and delete the ones that aren't large, giving us L_i. (this is the **counting step**).