**Recap of the last class:**

- In the last lecture, we learnt about two methods for generating candidates for frequent itemsets. Once we generate the candidates, we have to compute the support for each candidate. We also introduced hash tree for efficiently pruning candidates. A candidate *k*-itemset can be pruned if it has a (*k*-1)-subset that is not frequent. A hash tree can be used to perform this checking. We assumed that the expected size of each leaf in this tree is O(1). Let *u* be the degree of this tree. The height of this tree is (*k*-1). The expected size of each leaf will be $\frac{|F_{k-1}|}{u^{k-1}}$. We can choose a value for *u* such that this value is O(1).

**Expected time for candidates pruning:**

For each candidate in $C_k$, there are (*k*-1) subsets of size (*k*-1) each. For each subset we have to traverse through the hash tree to check if this subset is frequent or not. This will take O(*k*) time for each subset. This means that we spend O($k^2$) time for each candidate. Therefore, the total time spent on candidate pruning is $O(k^2|C_k|)$.

**Computing the support for candidates in $C_k$:**

We can compute the support for any *k*-itemset in O(*nk*) time, where *n* is the number of transactions in the database. Here we assume that each transaction is supplied to us as a *d*-bit array (where *d* is the number of possible items). Therefore, we can compute the support for all of the candidates in $C_k$ in a total of $O(nk\ |C_k|)$ time.

**Another way:**

For each transaction, increment the support by 1 of each candidate that is contained in the transaction. Use a hash tree to store all the candidates in $C_k$.

Here again we can ensure that the expected size of each leaf is O(1). If *u* is the degree of this hash tree, then the expected size of each leaf will be $\frac{|C_k|}{u^k}$. We can choose *u* such that $|C_k| = \Theta(u^k)$. In this case, a possible choice for the hash function at each node could be: $h(x)=x \bmod u$.

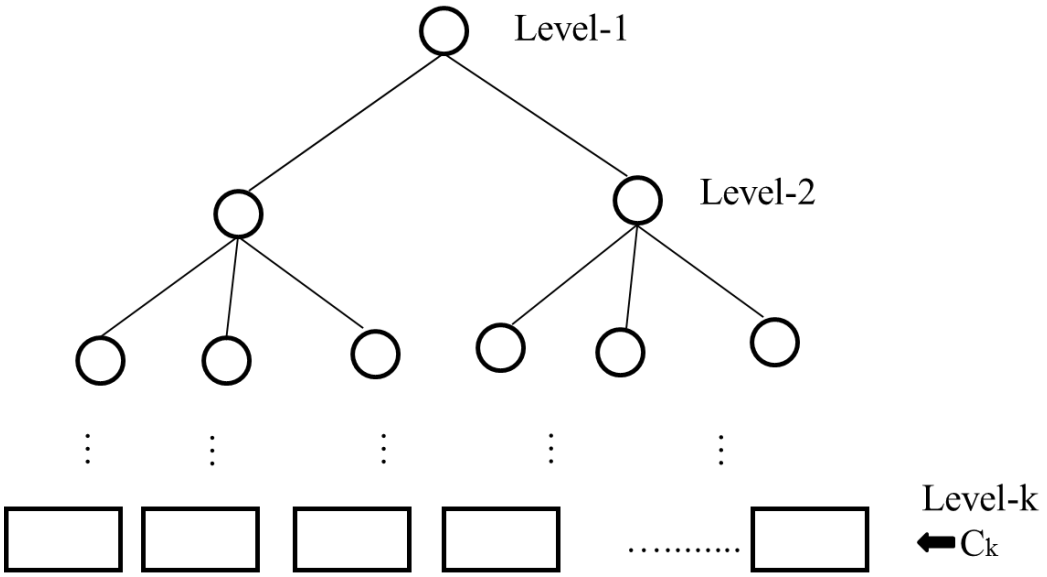


**Figure 1: A hash tree is used to store all the candidates in $C_k$.**

For each transaction $T$ do

For each $k-$subset $Q$ of $T$ do

Hash $Q$ into the hash tree. Once a leaf is reached,

For each candidate in the leaf

Increment the support by 1 if the candidate is the same as $Q$.

**Expected Run Time** $= O\left(\sum_{t_i \in DB} \binom{|t_i|}{k}.k\right)$. If $w$ is the maximum number of items in any transaction, then, this run time becomes $O\left(n\binom{w}{k}.k\right)$.

**Rules Generation:**

- Let $X$ be any frequent itemset. We generate rules from $X$ as follows:
  - Consider all non-empty and proper subsets $Y$ of $X$.
  - Compute the confidence of the rule $Y \rightarrow X - Y$

**Note:** Confidence for this rule is $\frac{\sigma(X)}{\sigma(Y)}$

- If $Y' \subset Y$ then $\sigma(Y') \geq \sigma(Y)$.

Therefore, (the confidence of $Y \rightarrow X - Y$) $\geq$ (the confidence of $Y' \rightarrow X - Y'$)

**Idea:** Use the above observation and a level wise approach. At level $k$ generate rules with $k$ items in the consequent, starting from $k = 1, 2, \cdots$

**Example:**

If the rule $\{abc\} \rightarrow \{d\}$ does not have enough confidence then we can ignore the following rules: $\{ab\} \rightarrow \{cd\}$, $\{bc\} \rightarrow \{ad\}$, and so on.

**Note:** We can generate candidates at each level just like in the case of frequent itemsets generation.

$\{ab\} \rightarrow \{cd\}$ can be chosen as a candidate if $\{abd\} \rightarrow \{c\}$, $\{abc\} \rightarrow \{d\}$ have enough confidence.

**A randomized rules mining algorithm (Toivanan 1996)**

**Basic idea:**

- Let DB be the given database
- Let $minSupport$ be the target support
- Pick a random sample $S$ from DB
- Identify itemsets from $S$ with a support of $\geq ms$ where $ms < minSupport$
- Let $Q$ be the collection of frequent itemsets in $S$
- For each itemset $q \in Q$ calculate its support in DB by examining the rest of the DB.
- Output each $q \in Q$ whose support in DB is $\geq minSupport$.

**Note:** If every frequent itemset in DB is also frequent in *S,* then we are in good shape. If *ms* is small enough, then this will happen with a high probability. Also note that all the itemsets that are output by the above algorithm are frequent. With some very low probability some of the frequent itemsets of DB may not be present in the output.

**Analysis:**

Let ***X*** be any itemset whose support in the DB is *f(**X**)*.

Let the support of ***X*** in the sample ***S*** be *f'(**X**)* and let $|\boldsymbol{S}| = s$.

*f(**X**)* and *f'(**X**)* are expressed as fractions.

Q: what can we say about *f'(X)* as a function of *f(**X**)*?

Note that the support for $X$ in the sample has a binomial distribution B($s$, $f(X)$). In the following analysis $\delta$ and $\delta'$ are user defined probabilities (and very small). For example, $\delta'$ could be $n^{-\propto}$, $n$ being the number of transactions in DB.

Using Chernoff bounds we have:

$Prob\left(sf'(X) < (1-\epsilon)sf(X)\right) \leq \exp\left[\frac{-\epsilon^2 sf(\boldsymbol{X})}{2}\right]$. We want this probability to be $\leq \delta$ so:

$$\delta \geq \exp\left[\frac{-\epsilon^2 sf(X)}{2}\right] \Rightarrow \ln(\delta) \geq \frac{-\epsilon^2 sf(X)}{2} \Rightarrow s \geq \frac{2\ln\left(\frac{1}{\delta}\right)}{\epsilon^2 f(X)}.$$

Let $N$ be an upper bound on the number of frequent itemsets in DB and $m$ denote *minSupport*, then:

$Prob(\exists$ a frequent itemset of DB not frequent in $S) \leq N \exp\left[\frac{-\epsilon^2 sm}{2}\right]$ (by the union bound).

We want this probability to be $\leq \delta'$ so:

$$\delta' \geq N\exp\left[\frac{-\epsilon^2 sm}{2}\right] \Rightarrow \ln\left(\frac{\delta'}{N}\right) \geq \frac{-\epsilon^2 sm}{2} \Rightarrow s \geq \frac{2\ln\left(\frac{N}{\delta'}\right)}{\epsilon^2 m} \blacksquare$$

We infer that all the frequent itemsets of the database DB will be output by the randomized algorithm with a probability of at least (1 - $\delta'$), if $s$ is chosen appropriately (as given in the above inequality).