Author: Lorenzo Epifani

Email: lorenzo.epifani@unisalento.it
lorenzo.epifani@polimi.it
lor.epi.job@gmail.com

This document has been extracted from my handbook with the purpose to enclose it in a git repository. My github:

`https://github.com/Lorenzo-Epifani`

The whole book will be available as soon as I finish it.

Subsection 1.4.6:

A-priori algorithm

A multi-step approach algorithm **A-Priori** optimize the usage of **main memory**. The idea is to take advantage of the **monotonicity** of the **frequency** of an itemset respect to the **number** of items in that itemset:

(1.4.10)Proposition: A-priori principle (Itemsets frequency monotonicity)

Given a generic **baskets** list B, and an **itemset** b, it holds:

$$\begin{aligned} \text{supp}_B(b) = s_b &\implies \forall b_{sub} \mid b_{sub} \subseteq b : \text{supp}_B(b_{sub}) \geq s_b \\ &\implies \forall b_{sup} \mid b_{sup} \supseteq b : \text{supp}_B(b_{sup}) \leq s_b \end{aligned} \tag{1.4.18}$$

As a consequence of this property, we can state that if an itemset is **frequent**, then **all of its subsets must also be frequent**.
Vice-versa, if an itemset is **not frequent**, then **none of its supersets can be frequent**.

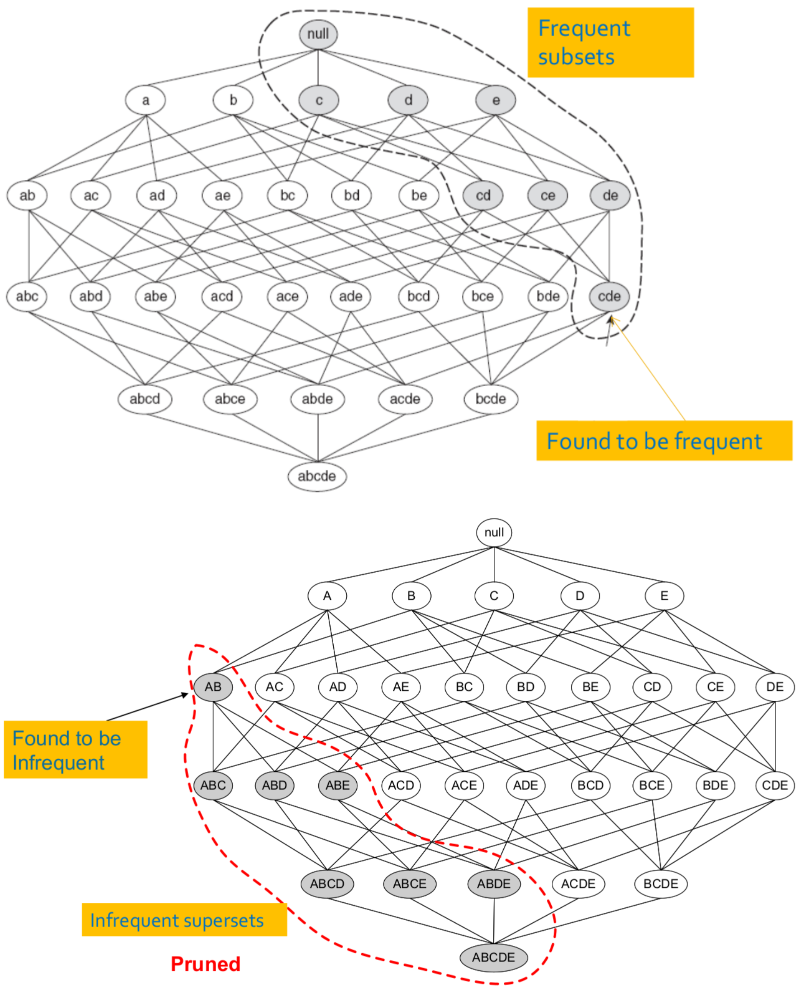

Figure 1.7: Illustration of the A-priori principle

Therefore, consider having **list** of **itemsets** $\mathbb{I}$, a list of **baskets** B and a frequency **threshold** f_t (itemsets with a **support** over $B \geq f_t$ are considered **frequent** in B), the **3** steps of the **A-priori** algorithm are described as follow:

- (1): **Initialisation**:
  We initialize $\mathbb{I}$ with the **universe** of all the possible **items** (i.e. itemsets with **cardinality** 1).

- (2) **Filter frequent:**
  Read baskets from B and count in **main memory** the occurrences of **each itemset** $\in \mathbb{I}$ in order to find **frequent** itemsets. This task requires only memory **proportional** to $|\mathbb{I}|$

- (3) **Generate $\mathbb{I}^*$:**
  In this step we **generate** a new **list** of itemsets $\mathbb{I}^*$ starting from the frequent ones of $\mathbb{I}$ found in the previous phase. Generation **criteria** are described separately. Go back to (2) using $\mathbb{I}^*$ as the new $\mathbb{I}$ looping

In the first iteration, $\mathbb{I}$ contains only **single** items, and the goal of the step (2) will be to find **frequent items**. Thus, at the end of the first iteration, $\mathbb{I}^*$ will be:

$$\mathbb{I}^* = \{\{a_i, a_j\} \mid \text{supp}_B(a_i) \geq f_t, \text{supp}_B(a_j) \geq f_t\} \tag{1.4.19}$$

That is, the set of all **possible pairs** made with the **frequent items**. With each **successive** iteration, the required **memory** of the algorithm **drops exponentially**. This means that the memory required by the step (2) of the **first iteration** has a bigger order of **magnitude** than the following **steps** of the following **iterations**. After the step (2) of the second iteration, we have all the **frequent pairs**. This result can be achieved with **A-Priori** without storing the **support** for each **possible pair** as we did in the previous algorithm (section 1.4.5).