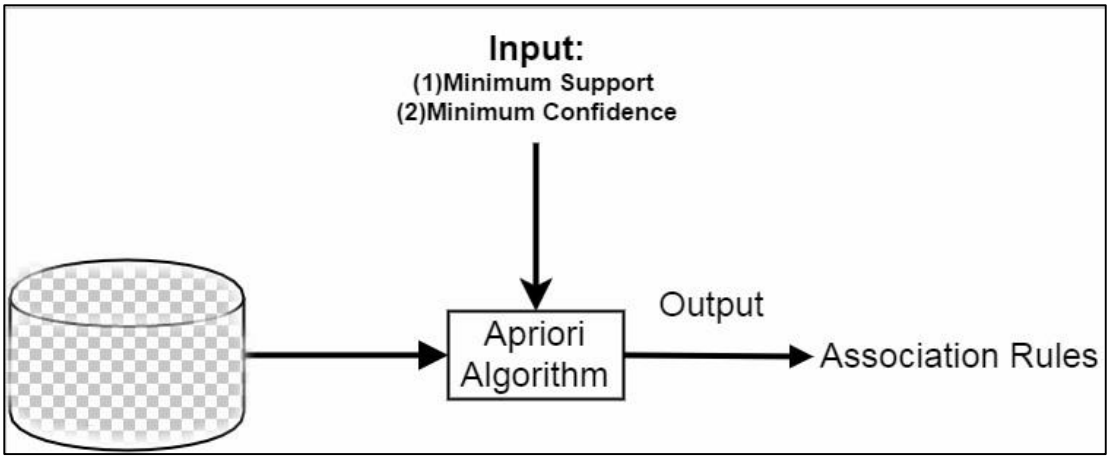
本 Project 使用 JAVA 程式語言實作原始的 Apriori 演算法，應用於 IBM Quest Synthetic Data Generator 產生出來的多個不同的資料集，找出資料裡的 Association Rule，最後將測試結果與 Weka 的結果進行比較，也與 hash-tree 與 FP-growth 技術的 Apriori 演算法進行比較，以及對比較的結果進行分析與討論，藉此思考此 Project 的優缺點以及可改進之處。

一、系統架構

- **架構**



圖一、系統架構圖

使用 IBM Quest Synthetic Data Generator 作為例子，並輸入兩個參數：(1) Minimun Support (2) Minnimum Confidence，再透過我們實作的 Apriori 和 FP-growth 演算法，找出所有符合條件的 Association Rule。

- **Output Format**

我們採用 ASCII 的輸出格式，格式內容如下：

- Each line contains a sequence
- The first number is the total number of Itemsets in this sequence
- After that, each Itemset is displayed, first by the number of items in this Itemset, then followed by the items in that Itemset, in increasing order.

範例：

Command: bin/seq_data_generator seq -ncust 0.003 -ascii -fname sample

Dataset:

5 1 2479 2 2154 5477 4 440 2276 6036 9838 4 6639 7926 9054 9748 1 1247 4 3 765 1542 7203 3 5854 7309 8827 1 5993 4 991 4956 6376 7993 6 1 9033 3 1845 7713 8778 3 1285 1705 7890 3 4049 6908 7443 2 765 1739 2 3627 9693

- **Apriori Algorithm**

Apriori Algorithm 資料探勘領域中，是用來找尋關聯式規則(association rules)的經典演算法之一。設計目的是為了處理包含交易內容的資料庫(例:

顧客購買的商品清單或者網頁常訪清單等)數據間的關聯式規則。通常會從 item 數為 1 的高頻率項目集合(Frequent Itemset)開始找起，每經過一回合要找的高頻率項目集合的 item 數就會加一，直到找不到為止。而回合的處理又分成第一回合跟第 k 回合。如果是第一回合，會把資料庫內所有的 item 都當成候選項目集合(Candidate Itemset)，利用最小支持度(Minimum support)判斷是否為高頻率項目集合。第 k 回合則是由(k-1)回合的高頻率項目集合產生 item 數為 k 的候選項目集合，再利用最小支援度(Minimum support)判斷哪些是高頻率項目集合。等到所有的高頻率項目集合都找到後，會將每個高頻率項目集合拆解成{A},{B}兩組集合，判斷在 A 發生情況下 B 發生的機率有沒有滿足最小信賴度(Minimum confidence)，如果滿足則成為其中一組 association rule。等到所有高頻率項目集合都判斷完，就可以產生資料庫當下的 association rules。

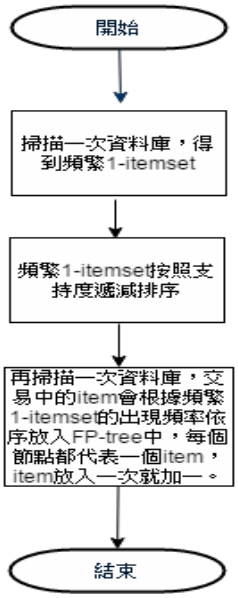
- **FP-Growth Algorithm**

原始 Apriori Algorithm 利用窮舉法把所有可能的組合都列出來，再刪除不滿足最小支持度的候選項目集合。但這種做法很沒有效率，除了會產生大量的候選項目集合，還會發生多次掃描資料庫的情形，這些都是 Apriori Algorithm 的瓶頸。FP-Growth 可以很好的緩解這些瓶頸，以下做簡單的介紹：

為了避免產生大量的候選項目集合，使用 Frequent-Pattern tree(FP-tree)結構壓縮資料庫，可以避免代價較高的資料庫掃描。此方法不用生成候選項目集合，並採用 divide and conquer 的策略，提供高頻率項目集合資料庫壓縮成的 FP-tree，但仍保留項目關聯資訊。

FP-tree construction

- 掃描一次資料庫，得到頻繁 1-Itemset
- 將項目按支持度遞減排序
- 再一次掃描資料庫，建立 FP-tree



由滿足最小支持度的 item 開始由下向上走訪 FP-tree，會與走訪到的節點 item 形成 Frequent Itemset，每多走訪一個節點，就多產生 item 數加一的 Frequent Itemset。直到所有滿足最小支援度的 item 全部走訪完，就可以得到所有的 Frequent Itemset。

- Advantage：不需要形成候選項目集合、使用壓縮的資料結構、避免重複掃描資料庫。
- Disadvantage：需要額外建立 FP-tree。

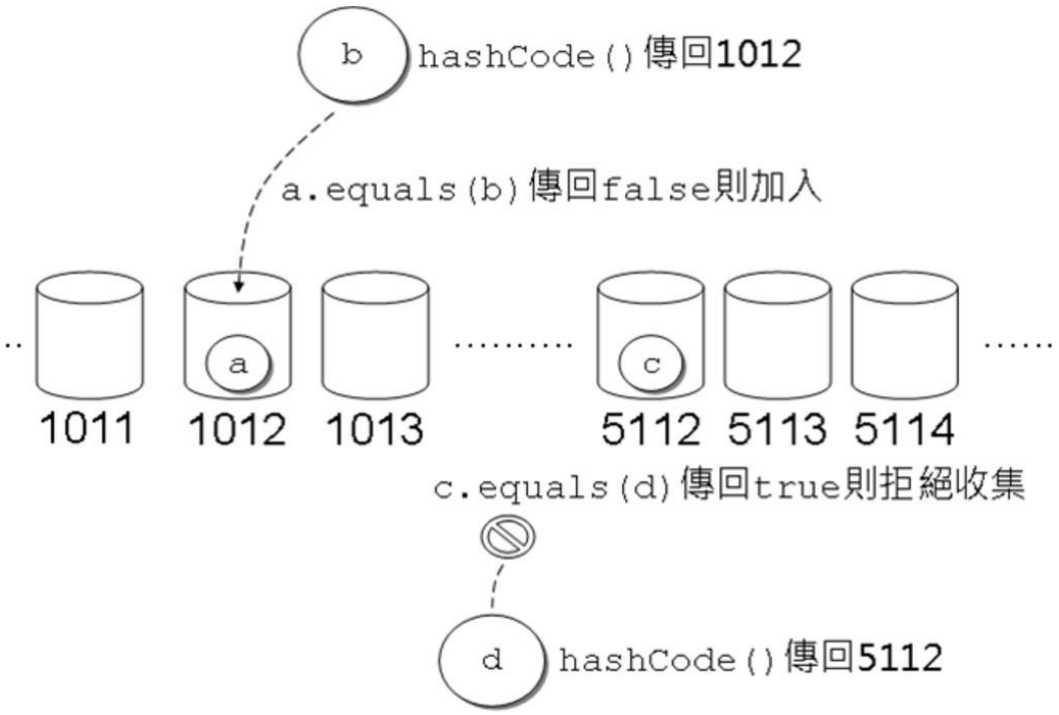
二、演算法

本 Project 採用 JAVA 實作 Apriori Algorithm，目的在於探討使用最直觀的暴力破解法後的效率與這些提升效率的技術的效能差別，進而探討其原因與改善方法，此節說明本 Project 的核心技術。

- **Union Operation**

我們實作了集合的聯集運算，此運算可用於判斷 Itemset 是否出現在另一個 Transaction 中，有助於計算 Itemset 的 Support 值和產生兩個 Itemset 的聯集結果來產生 Candidate Set 以及取得最後的 Frequent Itemsets。

我們採用了 JAVA 內的 HashSet 類別，此類別的特性是當有 Item 加入到此 HashSet 時，會先透過自行定義的 hash function 與 equal function 來判斷是否有相同的 Item 存在於集合中，避免相同的 Item 再次存入，以保證最後集合中的 Item 皆不重複，示意圖如圖二。



圖二、HashSet 運作示意圖

將兩個集合內的 Item 分別加入到一個 HashSet 中，即可產此兩個集合的聯集結果。

- **Candidate Set Generation**

First Candidate Itemset 由所有有出現在資料庫的不同 Item 組成，考慮到同一個 Item 可能出現在多筆交易中，所以並不能只是簡單的將所有交易內的 Item 加入到 Candidate Itemset，必須考慮到重複的問題，因此，我使用到了 HashSet 類別，根據此類別的特性可以輕易的取得所有不重複的 Item 所組成的集合，即 First Candidate Itemset。

第 k 個 Candidate Set 為第 k-1 個 Frequent Itemset 的元素所組成的所有含有 k 個 Item 的集合(k>1)，所以我們先將第 k-1 個 Frequent Itemset 內的 Item 兩兩作聯集運算，留下所有含有 k 個 Item 的集合，並使用 HashSet 避免產生相同的集合，得到第 k 個 Candidate Itemset。

- **Calculate Support Value**

計算 Itemset 的 Support 值必須逐一與資料庫內其它資料判斷此 Itemset 是否有出現，即是否被包含在其它的 Itemset 中，而我們採用聯集的特性來判斷集合的包含運算，若有一個 Itemset 1 被包含在另一個 Itemset 2 中，則此兩個集合的聯集結果必等於 Itemset 2，如下公式(1)所示。

$$A \in B \leftrightarrow (A \cup B) = B \qquad (1)$$

因此，我們只要將 Candidate Set 裡的 Itemset 兩兩進行聯集並判斷結果是否等於另一個 Itemset，就可以判斷是否有被包含，然後計算總共被多少 Itemset 所包含，即可取得 Support 值，並將小於 minimum support 的 Item 過濾掉，得到此一回合的 Frequent Itemset。

- **Rule generation**

當演算法取得最後的 Frequent Itemset，必須從 Frequent Itemset 產生所有可能的 Rule，並計算此 Rule 的 confidence 值，若大於 minimum confidence 值則留下，否則過濾掉。

假設目前有 Frequent Itemset L，若要產生所有的 Rule，必須先產生 L 的所有子集合 F，再產生$F \rightarrow L - F$ 的規則，若$|L| = k$，則可產生$2^k - 2$種不同的 Rule，於是我們採用 k-bits binary number 來對應到每一個子集合，考慮從$1,2,\ldots,2^k - 2$的每一個數字轉成二進制表示，進而產生所有 Rule，表一為$k = 3$的範例，其對應方式如下條列說明：

Binary number 的每一個位元對應原 Frequent Itemset 的 Item 的位置。若此位元$= 0$，則 Item 加入 F 中，反之，則 Item 加入$L - F$中。

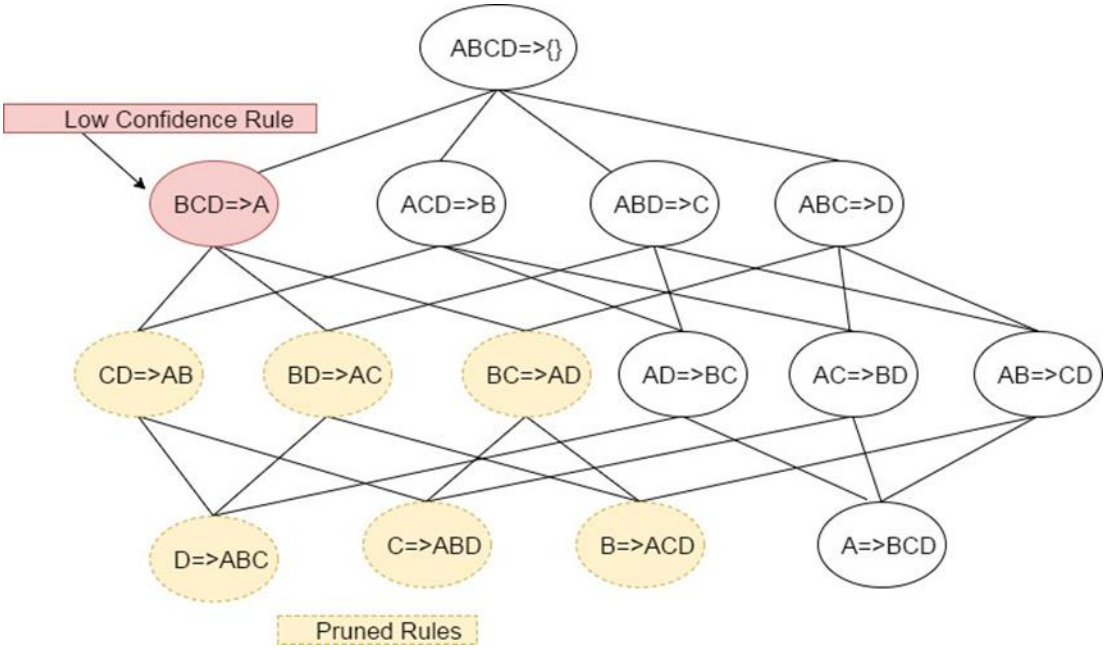
假設$L = \{a, b, c\}$：

Binary number	Rule
001	ab→c
010	ac→b
011	a→bc
100	bc→a
101	b→ac
110	c→ab

表一、k = 3之對應範例

若產生所有的 Rule 再進行過濾，可能會導致效率低落，因此，如何有效提前過濾掉不需要再考慮的 Rule 是重要的關鍵。

由於從同一個 Itemset 產生的 Rule 的 Confidence 具有 anti- monotone property 的性質，例如：$L = \{A, B, C, D\}$，$c(ABC \rightarrow D) \geq c(AB \rightarrow CD) \geq c(A \rightarrow BCD)$。藉由此性質，當我們發現一個 Rule 的 confidence 小於 minimum confidence 時，可以依照圖的規則提前過濾掉由同一個 Itemset 產生的其它 Rule，而我們發現若一個 Rule 2 是從在另一個 Rule 1 產生的，則 Rule 2 對應的數字與 Rule 1 對應的數字做 AND 運算，必等於 Rule 1，由此來實作此段說明的過濾法。



圖三、Rule 過濾示意圖

三、實驗結果

- **開發環境**

實驗環境：

實作語言	Java
CPU	intel-i5-3570 3.4GHz
Memory	8GB
OS	windows 10x64

實驗內容：

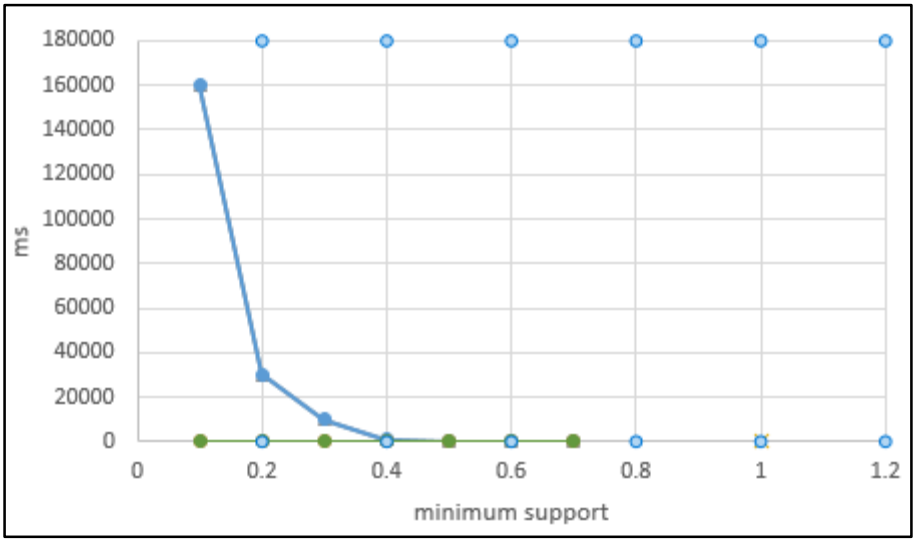
- 效能評估：執行時間(ms)
- 評估變數：Minimum Support, Minimum Confidence
- 比較演算法：Apriori Algorithm v.s. FP-Growth

實驗一

在此實驗會固定最小信心度(Minimum Confidence)為 0.6，藉由改變最小支持度(Minimum Support)來看其對各演算法的影響，以及各演算法的效率比較。

Min Support	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4
Rule 數量	357	118	53	27	21	9	9	6

表二、不同 Minimum Support 與產生的 Rule 數量



圖四、兩種演算法的效能差異（綠色 FP-growth，藍色 Apriori）

由上述實驗數據可以發現，隨著 minimum support 上升，中間產生的 Candidate Itemset 越來越少，Apriori Algorithm 的執行時間會有明顯的下降。原因是因為 Apriori Algorithm 在產生 Candidate Itemset 和用 Candidate Itemset 比較 minimum support 是用窮舉，很多迴圈的暴力的演算法。

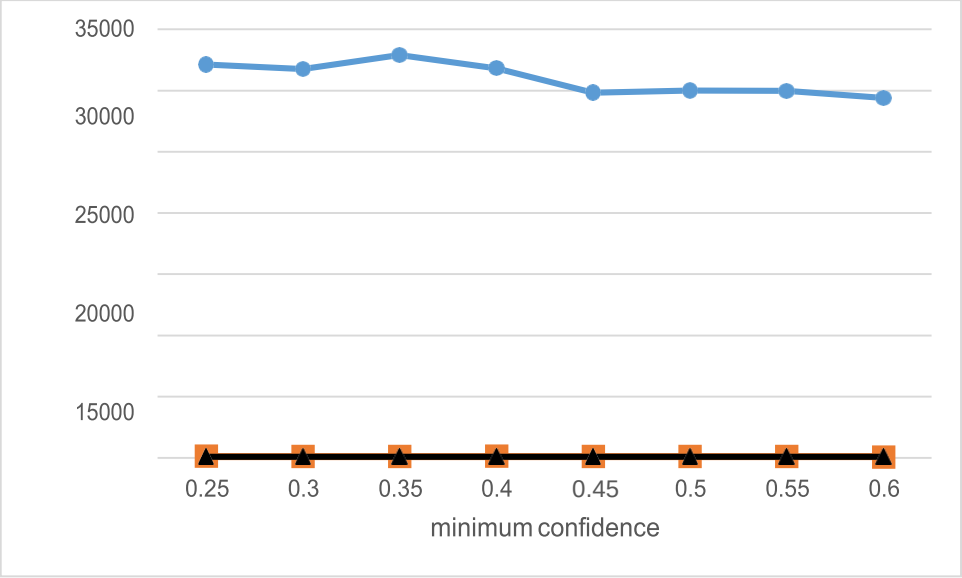
相較於 Apriori Algorithm，FP-growth 沒有受到太大的影響。FP- growth 則是因為不會產生 Candidate Itemset，所以不會受到影響。因為避免用暴力的方法計算，所以後者的效能明顯優於前者。最後可以發現，隨著 minimum support 越來越大，兩個演算法的執行時間都會收縮穩定，較大的 minimum support 可以過濾掉較多的 Candidate Itemset，節省了許多組合和比較的時間。

實驗二

在此實驗會固定最小支持度(Minimum Support)為 0.1，藉由改變最小信心度(Minimum Confidence)，來看其對各演算法的影響，以及各演算法效率比較。

Min Confidence	**0.25**	**0.3**	**0.35**	**0.4**	**0.45**	**0.5**	**0.55**	**0.6**
Rule數量	326	288	252	226	196	153	132	118

表三、不同 Minimum Confidence 與產生的 Rule 數量



圖五、兩種演算法的效能差異（橙黑 FP-growth，藍色 Apriori）

經測試，跑出的實驗結果與 weka 是一致的，由上述實驗數據可以發現，隨著 minimum confidence 上升，演算法的執行時間都沒什麼變化。原因是因為改變 minimum confidence 只會影響 Frequent Itemset 找 Association rule 的計算，而在這邊 Frequent Itemset 產生的數量都一樣而且很少，所以對效能的影響不大。

四、結論

- **資料集的格式與工具所讀取的格式不同**

  由於 IBM Data Generator 產生出來的資料格式是一連串數字所組成的文件，而 Weka 的分析工具是讀取 ARFF 格式的檔案，因此，前者產生出來的資料並不能直接使用於後者。

  解決方法：

  根據 ARFF 格式寫了一個格式轉換器，將 IBM Data Generator 轉換成 ARFF 格式的檔案，透過轉換器我們可以應用同一個資料集於網路上常見的資料探勘工具。

- **Rule 的產生方法**

  演算法最後的步驟是由 Frequent Itemset 產生所有可能的 Rule，並判斷其 confidence 是否大於 minimum confidence，要產生所有 Rule 就必須先產生 Frequent Itemset 的所有子集合，再組合成所有可能的 Rule，但要如何才能快速的找到一個集合的所有子集合，並取得其對應的 Rule，以及當發現一個 Rule 被淘汰時，如何提前過濾掉其它不需要再考慮的 Rule。

  解決方法：

  一個含有有 k 個元素的集合，我們使用$1,2,\ldots,2^k-2$的二進制標記法

來產生所有可能的 Rule，並將被淘汰的 Rule 記錄下來，透過 AND 的運算其它 Rule 是否被包含在此被淘汰的 Rule，藉此提前過濾掉不需考慮的 Rule。

- **心得：**

在一開始，對於資料量大一點的處理是一點頭緒也沒有，Big Data 是近年來很紅的議題，而實際上卻不知道有了這些數據到底能做什麼？這次的 project 讓我瞭解到找關聯便是應用這些資料的方向之一，實現 Apriori algorithm，能練習找關聯規則的必備概念。

經過這次作業，讓我更加瞭解 data mining 中找 Association rule 的困難處和解決方法的細節。FP-Growth 避免產生 Candidate itemset，所以大大的改善了效能。儘管 Apriori algorithm 效能差，還是有值得學習的細節，像這次實作時產生 Candidate itemset、利用 Candidate itemset 產生 Frequent itemset 和利用 Frequent itemset 產生 Association rule 的部分，改善效能的方法是值得學習的，那些結構和處理的方法的概念，感覺可以運用在不同 data mining 的題目，甚至使用在不同領域的題目。

這個作業仍有許多可以改善的空間，我並沒有實現 hash-tree 算法找關聯規則，雖然使用的是窮舉法，但也因此深刻的認識了此演算法的每一個步驟，也了解了 hash-tree 與 FP-Growth 等改善效能的方法其優缺點，同時學會了網路上許多資料探勘的工具使用方法，過程中雖然辛苦但充實的。