有效率的挖掘高頻物項集合演算法

An Efficient Mining Algorithm of Frequent Itemsets

楊東麟 (Don-lin Yang)
逢甲大學資訊工程研究所
(04)24517250-3700
dlyang@fcu.edu.tw

楊文昇 (Wen-sheng Yang)
逢甲大學資訊工程研究所
(04)23596073
wensheng@mail.tahhsin.com.tw

摘 要

我們針對資料挖掘(Data Mining)中發掘關聯式規則(Association Rules)的程序，提出新的架構。本研究是在前製處裡中，將相同交易物項集合做合併，累計交易次數，減少資料筆數，並利用由上而下循序處裡，區分高頻物項集合(Frequent itemsets)與非高頻物項集合(Infrequent itemsets)，可以有效率發掘關聯式規則，在過程中不需要產生候選物項集合，並對資料庫做一次的交易次數資料合併處理及一次的 I/O 掃描即能產生高頻物項集合。我們的結果在支持度變化中執行效率好而且呈現穩定狀態，而 Apriori 和 DHP 演算法的執行效率受支持度的影響非常大，尤其支持度較小時，有大量的高頻 2-物項集合結合為候選 3-物項集合，需要更大量的主記憶體，會直接影響演算法的進行，而我們的方法證實能有效減少主記憶體的使用，較不受支持度變化而影響效能。

關鍵字:資料挖掘、關聯式規則、交易物項、高頻物項集合、非高頻物項集合

We proposed a new structure for finding association rules in data mining. The Apriori method is very time consuming. It joins Items and accesses databases repeatedly. It also requires large amount of main memory and is very slow in execution. This research combines the same Items, accumulates the number of items, reduces data items to be counted, and uses top-down process to separate frequent itemset and infrequent itemset. It can mine association rules more efficiently. There is no need to produce candidate itemsets, and it is able to produce frequent itemsets in one database scan.

This method provides more efficient execution than Apriori or DHP does. The support value has less impact on our method, but Apriori and DHP are effected greatly by support value, especially when the support is small, more main memory and execution time are needed. Our method has been proved to use less main memory and execution time, and is less sensitive to the variation of support values.

1. 簡介

資料挖掘或知識探勘(Knowledge Discovery)[7]是藉由自動或半自動的機制在大量的資料中尋找有意義的規則或型態的一種分析方法，在往常，要在資料庫中尋找資料的特性或分析結果是藉由資料庫軟體的協助，如使用 SQL 語法查詢或藉由 OLAP(On Line Analytic Processing)與資料倉儲(Data Warehousing)[15]的運用找出對於資料的分析

結果，而資料挖掘(Data Mining)對於資料的分析與模型的建立是一種有效的工具;以技術面來看，他泛指各式各樣從大量資料中歸納出資訊與知識的方法，其中包含了關聯式法則、時間序列、Classification rule、 Clustering rule 等。

關聯式法則主要被利用來尋找資料庫中項目或屬性之間共同發生的關係，例如找出商場中某些商品所具有共同被購買的關係，像"80%購買牛奶的人，也會同時購買麵包"[1]，當決策者取得這些資訊後，就可以考慮針對這兩項商品的銷售做改良，也許是將牛奶和麵包的展售櫃檯擺在一起，也可能針對某種麵包和牛奶做促銷。所以利用關聯法則挖掘的目的，是希望能從資料相對發生次數的分析著手，找出未知的關係，作為決策時的參考。

目前挖掘關聯式法則，有 SETM、AIS[1]、Apriori、AprioriTid、AprioriHybrid [2]、雜湊法(Hash-Based)[5][6]、Partition[8]、DIC[9]、Column-Wise[10]、 MaxEclat、MaxClique[11][12]、Multiple-level[13] 、FS、SS[14]等演算法，是依據 Apriori 演算法為基礎的各種演算法，他們都從單一物項開始，先利用一些演算法找出所有可能的候選物項集合，再從這些候選物項集合找出真正的高頻物項集合，而下一階段的開始就是將上一階段的高頻物項集合的排列組合，再重新分析，往後舊如此循環下去，所以當重覆 n 次後，就會出現 n+1 個高頻物項集合。無法擺脫產生大量的候選物項集合，佔用非常龐大的主記憶體，而造成主記憶體的空間不足，而無法進行關連聯式法則挖掘或需多次掃描資料庫，I/O 次數非常頻繁，浪費挖掘時間。

我們的研究著重在簡化候選物項集合產生與減少重覆掃描資料庫，而能有效的產生高頻物項集合。在進行關連聯式法則挖掘前，須先進行前製處裡，將相同交易物項集合做合併，累計交易次數，減少資料筆數，若交易筆數發生相同的物項集合平均重複筆數有 n 筆，合併後資料筆數將縮為(交易筆數*1/n)，I/O 次數將為(交易筆數*1/n)次，並利用由上而下循序處裡，區分高頻物項集合與非高頻物項集合，可以有效率發掘關聯式規則，在過程中不需要產生候選項目集合，並能在一次的前置處理及一次的 scan 資料庫即能產生高頻物項集合。

2. 相關研究

在對資料庫作關聯式法則挖掘，有許多的研究提出很多的技術和方法，而很多的研究是以 Apriori 演算法為基礎再延伸或推導新的演算法，主要是由於 Apriori 演算法的步驟簡單，且又有系統實作容易的優點，但其缺點有執行效率不佳的問題。

在此針對一般的關聯式法則做一個正式的定義 Apriori[2]，假設一商品物項集合 I(itemset)包含了所有可能的商品物項{i1，i2，…im}，並設 D 為一群商品交易紀錄的集合，且每一筆交易紀錄 T(transaction)所包含的就是一群物項的集合，所以所有的 T 出現的物項都是可以被 I 所涵蓋的，而不管該物項的數量。一個關聯式法則的形成為前提物項集合(antecedent itemset) $\rightarrow$ 結果物項集合 (consequent itemset)，前後兩種物項集合都是 I 的子集合，且兩者的交集為空集合，對於一個關聯式法則為 $X \rightarrow Y$，X、Y 為兩個包含於 I 的非空集合，則支持度(support)是 D 中包含了 $X \cup Y$ 的交易所佔百分比。信賴度(confidence)是 D 中同時包含 XY 之交易數和包含 X 之交易數的比值。(支持度與信賴度都是介於 0 與 1 之間)

$$\text{支持度} = \frac{\#\text{of transactions which contain } X \cup Y}{\#\text{of transactions in the database}}$$

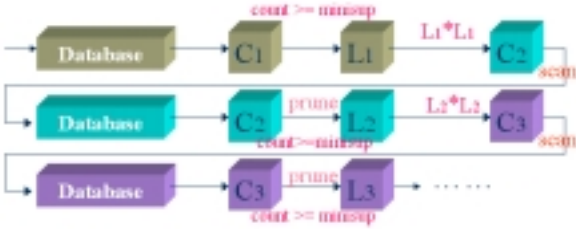
$$\text{信賴度} = \frac{\#\text{ of transactions which contain } X \cup Y}{\#\text{ of transactions which contain X}}$$

一個有效的關聯式法則，必須滿足"信賴度大於等於使用者預設最小信賴度 C 且支持度大於等於使用者預設最小支持度 S 的關聯式法則" [3]。而對於一個物項集合，我們定義其支持度為包含該物項集合的交易個數。高頻物項

集合(frequent itemset 或 large itemset)為支持度大於等於使用者預設最小支持度 S 乘以交易總數｜D｜的物項集合。例:若{XY}是一個高頻物項集合且{XY}的支持度除以{X}的支持度≥C 則 X⟶Y 是一個有效的關聯式法則。

2.1 Apriori algorithm

對於關聯式法則的挖掘，很多的研究報告常以 Agrawal *et al.* 於 1994 年提出的 Apriori 演算法[2]為基礎，它利用簡單且循序漸進的方式，找出資料庫中項目的關係，以形成規則。此方法是目前研究關聯式法則的入門演算法，可說是研究關聯式法則時最具代表性的演算法之一。



圖一. Apriori 演算法的演算程序

如圖一 Ariori 演算法描述如下:

(1) 首先訂定過濾規則強度的門檻值—最小支持度及最小信賴度。

(2) Apriori 演算法使用了候選物項集合(Candidate Itemset)的觀念，首先產生出物項集合，稱為候選物項集合，若候選物項集合的支持度大於或等於最小支持度(minisup)，則該候選物項集合為高頻物項集合(Large Itemset)。

(3) 在 Apriori 演算法的過程中，首先由資料庫讀入所有的交易，得出候選單物項集合(Candidate 1-itemset)的支持度，再找出高頻單物項集合(Large 1-itemset)，並利用這些高頻單物項集合的結合，產生候選 2 物項集合(Candidate 2-itemset)。

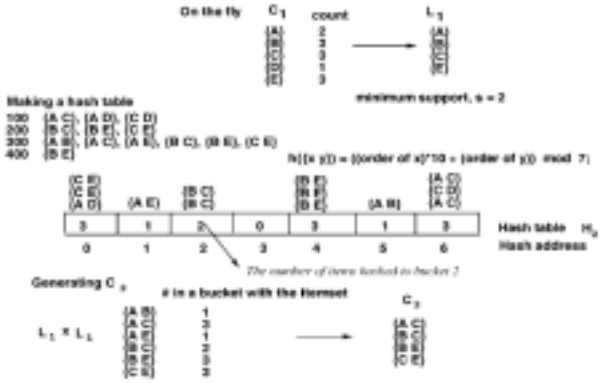
(4) 再掃描資料庫，得出候選 2 物項集合的支持度以後，再找出高頻 2 物項集合，並利用這些高頻 2 物項集合的結合，產生候選 3 物項集合。

(5) 重覆掃描資料庫、與最小支持度比較，產生高頻物項集合，再結合產生下一級候選物項集合，直到不再結合產生出新的候選物項集合為止。

存在 k 種物項的物項集合稱之為 k-物項集合(k-itemset)，令 C_k 表示有 k 個物項的候選物項集合(或稱為候選 k-物項集合)所組成的集合，L_k 表示有 k 個物項的高頻物項集合(或稱為高頻 k-物項集合) 所組成的集合，則用遞回方式產生候選物項集合的過程可以表示為：$C_{k+1} = L_k * L_k = \{X \bigcup Y \mid X, Y \in L_k, \mid X \bigcap Y \mid = k-1\}$，利用 Apriori_gen 函數先將兩個高頻 k-物項集合，組成一個候選(k+1)物項集合中，是否每一個 k-物項集合的子集合都是高頻物項集合，若不是則刪去(prune)該候選物項集合。

2.2 DHP Algorithm

在利用 Apriori 演算法作資料挖掘時可以發現，因計算物項過多造成執行效能緩慢，此瓶頸主要的原因在於高頻物項集合產生過多的候選物項集合，由其是候選 2 物項集合的情況最為嚴重。Chen *et al.* 於 1995 年提出了 DHP(Direct Hashing and Pruning)演算法[4]，主要利用了刪減不必要候選物項集合的概念來改善以往在關聯式法則挖掘時的效率，DHP 演算法是以 Apriori 演算法為基礎，另外再加入了 hash table 的架構，使其效能進一步的有效提昇。



圖二 DHP 利用雜湊法產生候選物項集合 C2(Park，*et. Al*，1995)

圖二是由 Ming-Syan Chen 等人於 1995 年提出 DHP 演算法[4][5]，是以 Apriori 演算法為基礎而於改善。在 Apriori 演算法做重覆資料庫掃描，計算候選項集合 C*k* 的 count 與支持度做比較，產生高頻物項集合 L*k*，而 DHP 主要的貢獻在於利用雜湊函數(hash function)過濾非高頻候選物項集合，然後再掃描資料庫計算未過濾的高頻候選物項集合，以減少候選物項集合的數目來掃描資料庫，DHP 演算法過濾非高頻候選物項集合是使用一種雜湊函數，並根據雜湊表(hash table)中 bucket 的數值去刪除不必要的候選物項集合，同時 DHP 會隨著回合數(iteration)的增加，而需要掃描的資料量會變小，其優點在於以雜湊表中 bucket 的數值當作刪除候選物項集合的上限，在第二回合時候可以大幅縮小 C2，因為要不斷計算雜湊函數，刪除不需要之候選物項集合，要掃描的資料量減少，因而提高執行的效率。

3. 研究方法

多數關聯式法則的演算法都以 Apriori 為基礎延伸產生，而 Apriori 執行過程中會產生大量的候選物項集合，造成主記憶體空間不足並多次掃描資料庫，I/O 次數頻繁，浪費挖掘時間，STD 演算法為了避免在執行中佔據太多的主計憶體及挖掘時間浪費，考慮如何減少產生大量的候選物項集合及資料庫掃描次數，能有效的提昇執行的速度，因而設計另一種演算法改善 Apriori 演算法的缺失，STD 演算法的概念是依據交易物項集合發生的次數判斷是為高頻物項集合或非高頻物項集合，並使用 Top-Down 資料處理的架構，與最大高頻物項集合其所有的子集合也必為高頻物項集合等概念，研究產生的演算法。

在關連聯式法則的執行，演算法相當耗費時間，而法則(rules)較容易產生，因此我們的研究重點在於演算法的效率上。

3.1 詞彙說明

在說明詳細的設計流程之前，對本研究的一些專有詞彙做說明：

- DB (Data Base):交易紀錄資料庫，包含所有的商品交易，每一筆的紀錄代表一次的交易，以 TID 來識別交易，而每一筆的交易中又包含了各種的交易物項集合(itemsets)
- TID (Transaction ID) :交易紀錄資料庫中用來代表每一筆交易的唯一代表
- DB1:交易紀錄合併資料庫，包含所有的商品交易，而每一筆為資料庫中相同物項集合的合併資料，以 TID1 來識別合併後交易資料
- TID1:合併後交易紀錄資料庫中用來代表每一筆物項集合及發生次數(frg)
- RS:表自 DB1 讀入資料記錄(data record)
- frg:表有相同交易物項集合的交易發生次數
- min-support:使用者自訂的最小發生次數
- Frequent Buffer:高頻緩衝區，此區存放高頻物項集合
- Infrequent Buffer:非高頻緩衝區，此區存放非高頻物項集合
- Temp Buffer:暫存緩衝區，此區存放非高頻物項集合
- Frequent itemsets:高頻物項集合，此集合必須符合 min-support 的要求
- Infrequent itemsets:非高頻物項集合，此集合未符合 min-support 的要求

3.2 STD 演算法的實作

為了能讓 STD 演算法的執行能有高效率，將 STD 演算法分為兩部份，第一部份是將 DB 做前置處理，做物項集合的排序及相同物項集合的交易資料做合併且紀錄交易次數，後再依物項集合的物項個數排序產生 DB1，第二部份以 DB1 依資料順序逐筆處理，判斷物項集合交易的次數是否滿足最低支持度，若是則物項集合為高頻物項集合，將此交易物項集合放入高頻緩衝區 (Frequent Buffer)，若不是則物項集合為非高頻物項集合，將此物項集合放入非高頻緩衝區 (InFrequent Buffer)，依此演算法執行，在節省 memory 及減少 I/O 方面得到非常好的效果。此外 STD 演算法有一特性可幫助我們 STD 演算法執行時，快速檢測是否為高頻物項集合，使高頻物項集合能快速的產生。

特性:假如物項集合是高頻物項集合，所有的子物項集合(subitemset)必定是高頻物項集合，因此不需要再進一步的檢測[6]。

3.2.1 前置處理

在進行資料挖掘前我們需要經資料整理，選取的流程，在此階段需要花費一些的時間成本，以避免不正確或無關緊要的資料進入資料挖掘的程序，確保資料在進入挖掘時是我們所需要的資料，因此為了讓資料挖掘更有效率進行，我們在此階段將資料作更進一部的整理，讓資料的筆數大幅減少且產生符合 STD 演算法的資料結構，對於進行我們的方法有非常大的幫助。

將 DB 交易資料庫中每一筆物項集合依物項編號由小至大排序，後依物項集合的物項個數由長至短做排序，把相同物項集合做 count 累加，代表物件集合交易發生次數，最後產生新的 DB_1，DB_1 內資料項有物項集合、TID_1，交易發生次數(frq)。產生新的 DB_1 後進入 STD 演算法處理。此部份資料的前置處理利用 sorting 功能以最短時間，能大量減少資料筆數，若資料筆數發生相同的物項集合平均重複筆數有 n 筆，合併後資料筆數將縮為(資料筆數 * 1/n)，I/O 次數將為(資料筆數 * 1/n)次，將大幅改善執行的速度。若平均重複筆數值越大，合併後的資料筆數越少，在進行資料挖掘的速度相對越快速，反之則慢。

3.2.2 找出所有的最大高頻物項集合

經過前置資料整理後，資料已符合 STD 演算法的資料結構，每筆資料內容含有交易物項集合、交易次數、TID_1，我們依此 3 項資料進行我們的演算法。

STD 演算法的重心在於推導出最大高頻物項集合，而在推導最大高頻物項集合的過程中，主要分為 5 個步驟概述如下：

(1) 在第一筆資料時，判斷交易次數≥支持度，為高頻物項集合，物項集合放入高頻緩衝區，若交易次數≤ 支持度，為非高頻物項集合，將物項集合放入非高頻緩衝區。讀次一筆資料後進入(2)，直到非高頻緩衝區有物項集合，進入(3)。

(2) 讀入第二筆以後(含第二筆)資料，物項集合到高頻緩衝區中判斷是否為高頻物項集合，若是，讀取次一筆資料後進入(2)，若為非高頻物項集合，進入(3)。

(3) 讀入資料的物項集合的物項個數是否小於非高頻緩衝區中的物項集合的物項個數，若小於，進入(4)進行非高頻緩衝區中將物項集合做分解處理，分解為更小物項個數的子物項集合，若物項個數相等則判斷資料的物項集合是否存在於非高頻緩衝區中。

   a. 存在 ⟹ 資料的物項集合的交易次數與非高頻緩衝區中的物項集合的交易次數相加，累加後的交易次數≥支持度，表示資料物項集合為高頻物項集合，放入高頻緩衝區中且刪除非高頻緩衝區中的

物項集合。如果累加後的交易次數< 支持度，為非高頻物項集合，更新非高頻緩衝區中的該物項集合的交易次數值為累加後的交易次數值。

b. 不存在 ⇒ 資料的物項集合的交易次數 ≥ 支持度，為高頻物項集合，放入高頻緩衝區中。若交易次數< 支持度，為非高頻物項集合，放入非高頻緩衝區中，並記錄 TID_1 值，做為分解子物項集合時判斷交易次數是否要相加之依據(在分解物項集合為子物項集合時，相同 TID_1 值交易次數不做相加，不相同 TID_1 值交易次數做相加)。

(4) 做分解子物項集合處理，將非高頻緩衝區中物項集合的物項個數(l)分解為(l-1)物項個數的子物項集合直到物項個數等於資料的物項集合的物項個數，進入(5)做高頻物項集合的判斷，分解過程如下:

a. 非高頻緩衝區中每個物項集合的物項個數(l)分解為子物項集合的物項個數(l-1)並放入暫存緩衝區(Temp Buffer)中，當 l 物項個數的物項集合分解為(l-1) 物項個數的子物項集合會產生 l 個子物項集合(含交易次數及 TID_1)，在分解過程中會有相同的子物項集合產生，不同的 TID_1 且有相同子物項集合的交易次數累加，而相同的 TID_1 且有相同子物項集合不作處理保留原資料，以減少產生子物項集合數目。

b. 產生的子物項集合放入暫存緩衝區時，判斷暫存緩衝區是否存在子物項集合。

b1. 不存在 ⇒ 將分解的子物項集合及交易次數、TID_1 放入暫存緩衝區中。

b2. 存在 ⇒ 與暫存緩衝區的物項集合的 TID_1(此物項集合曾經發生的 TID_1)做判斷是否相同，其中有一個 TID_1 相同，不做進一步處理(表來源相同，交易次數已做累加)，讀取次一筆資料。若與 TID_1 全不相同，分解的子物項集合的交易次數與暫存高頻緩衝區中子物項集合的交易次數相加，後更新暫存緩衝區中子物項集合的交易次數值並記錄分解的子物項集合的 TID_1。

(5) 此時已經完成非高頻緩衝區中每個物項集合的物項個數(l)分解處理，產生的子物項集合放在暫存緩衝區中，接下來判斷暫存緩衝區中的物項集合是否為高頻物項集合。

a. 讀取暫存緩衝區中的物項集合，到高頻緩衝區中判斷是否為高頻物項集合。

b. 若是為高頻物項集合(暫存緩衝區的物項集合被高頻緩衝區中的高頻物項集合包含) ⇒ 刪除暫存緩衝區的物項集合，並繼續讀取暫存緩衝區次一筆的物項集合。

c. 若不是高頻物項集合(暫存緩衝區的物項集合不被高頻緩衝區中的高頻物項集合包含)⇒再判斷暫存緩衝區的物項集合的交易次數值，若交易次數 ≥ 支持度，為高頻物項集合放入高頻緩衝區中，若交易次數< 支持度，為非高頻物項集合放入非高頻緩衝區中。

d. 將暫存緩衝區的該筆物項集合刪除掉。

e. 再繼續讀取暫存緩衝區次一筆資料(直至結束)，回到(3)。

以上為 STD 演算法的處理 5 個步驟。

3.2.3 範例說明

假設資料庫中有 6 筆交易紀錄資料，每一筆交易紀錄中各有若干個物項，其中各物項我們分別以 A,B,C,….等符號來表示。如表 1 所示。

在進入 STD 演算法前，交易資料庫(D)必須做前置處理，將各筆交易資料的物項集合編號由小至大與物項個數由長至短做排序，後將排序後相同物項集合做合併且紀錄交易次數

及 TID_1，產生新資料庫(D_1)，經處理後資料筆數為 5 筆，較原資料庫筆數少 1 筆資料，TID 6 的{BCE}合併至 TID_1 3，如表 2 表示新資料庫資料內容及排序。frq 表相同交易物項集合交易次數。

表 1. 交易紀錄資料

TID	Itemset
1	ACD
2	ABCE
3	BCE
4	BE
5	AF
6	BCE

表 2.交易次數合併後交易紀錄資料

TID_1	itemset	frq
2	ABCE	1
1	ACD	1
3	BCE	2
4	BE	1
5	AF	1

以下將利用一個簡單的例子來說明本研究在物項集合產生高頻物項集合的過程，我們設定最小支持度值為 2 次;表 1 為交易紀錄原始資料。當交易紀錄原始資料庫經過前置處理後產生(表 2)合併交易資料紀錄資料庫(D_1)，以 D_1 資料庫讀取資料第一筆資料{ABCE}，此筆資料的 frq 為 1，小於支持度，放入非高頻緩衝區。如表 3 所示。

表 3.{ABCE}處理後產生在 buffer 中的結果

Infrequent buffer	Frequent buffer
ABCE 1	

接下來，讀取第 2 筆資料{ACD}，{ACD}的 frq 為 1，先到高頻緩衝區檢測，高頻緩衝區內無高頻物項集合，又 frq 小於支持度(2)，此時非高頻緩衝區的非高頻物項集合的物項個數為 4，{ACD}的物項個數為 3，小於非高頻緩衝區的非高頻物項集合的物項個數，表示接下來的交易資料已無物項個數為 4 的資料，而{ABCD}是非高頻物項集合，因此{ABCE}必須做分解處理，才能在其子物項集合中找出可能的高頻物項集合，由{ABCE}經分解產生{ABC}，{ABE}，{ACE}，{BCE}子集合，每個子集合的 frq 各為 1，TID_1 為 2，並存放於暫存緩衝區(Temp Buffer)並刪除非高頻緩衝區中的{ABCE}，再判斷暫存緩衝區中每個子集合是否為高頻物項集合，先到高頻緩衝區檢測是否為高頻物項集合(若為高頻物項集合的子集合亦為高頻物項集合)，此時高頻緩衝區無高頻物項集合，{ABC}，{ABE}，{ACE}，{BCE}皆非高頻物項集合，又 frq 皆為 1 小於支持度，最後將{ABC}，{ABE}，{ACE}，{BCE}放入非高頻緩衝區中，並刪除暫存緩衝區子集合，如表 4 所示。

表 4. 經分解後 infrequent buffer 的結果

Infrequent buffer	⇒	Temp buffer	⇒	Infrequent buffer
ABCE 1		ABC 1，ABE 1 ACE 1，BCE 1		ABC 1，ABE 1 ACE 1，BCE 1

接下來判斷{ACD}是否存在於非高頻緩衝區中，不存在且 frq 為 1，將{ACD}放入非高頻緩衝區中，如表 5 所示。

表 5. {ACD}處理後的結果

Infrequent buffer	Frequent buffer
ABC 1，ABE 1，ACE 1， ACD 1	BCE

再到 D_1 資料庫讀第 3 筆資料{BCE}，{BCE}的 frq 為 2，到高頻緩衝區做檢測，{BCE}不存在高頻緩衝區中，但存在非高頻緩衝區，與非高頻緩衝區的{BCE}的 frq 相加，結果為 3，滿足支持度(2)，將{BCE}放入高頻緩衝區，並刪除非高頻緩衝區中的{BCE}，如表 6 所示。

表 6. {BCE}處理後的結果

Infrequent buffer	Frequent buffer
ABC 1， ABE 1， ACE 1， BCE 1， ACD 1	

再到 D_1 資料庫讀第 4 筆資料{BE}，{BE}的 frq 為 1，到高頻緩衝區做檢測，為高頻物項集合，因{BE}為高頻物項集合{BCE}的子集合，繼續讀第 5 筆資料{AF}，{AF}的 frq 為 1，到高頻緩衝區做檢測，{AF}不存在高頻緩衝區中，此時非高頻緩衝區的非高頻物項集合的物項個數為 3，{AF}的物項個數為 2，小於非高頻緩衝區的非高頻物項集合的物項個數，表示接下來交易資料已無物項個數為 3 的資料，因此{ABC}， {ABE}，{ACE}，{ACD}必須做分解處理，在分解過程中產生{AB}，{AC}，{BC}，{BE}，{CE}，{AE}，{AD}，{CD}，而{BC}，{BE}，{CE}是高頻物項集合{BCE}的子集合，故做刪除處理， {AB}，{AE}，{AD}，{CD} 的 frq 為 1，為非高頻物項集合，另{AC}的 frq 為 2，在 TID_1 的 1，2 發生，{AC}為高頻物項集合，如表 7 所示。

最後判斷{AF}是否存在於非高頻緩衝區中，不存在且{AF}的 frq 為 1，將{AF}放入非高頻緩衝區中，如表 8 所示。

最後資料執行結束，同時產生高頻緩衝區中的物項集合且都是最大高頻物項集合。

表 7. {AF}讀進後 infrequent itemsets 做分解的結果

Infrequent buffer	Frequent buffer
AB 1， AE 1， AD 1， CD 1， AF 1	BCE， AC

表 8.最後產生 Frequent 和 Infrequent itemsets 的結果

Infrequent buffer	Frequent buffer
AB 1， AE 1， AD 1， CD 1	BCE， AC

4. 實驗設備及實驗說明

為了驗證我們研究所提演算法之可行性及執行效率，我們利用以下的環境來進行挖掘效率的測試：

(1) 實驗平台

(1.1) PC : Pentium III 處理器 933 MHz

(1.2) 記憶體 : 384 Mbytes，硬碟空間 30 Gbytes

(1.3) 作業系統 : Windows 98

(1.4) 存取實驗資料庫的資料庫系統為 MS SQL

(2) 實作之程式語言：利用 Java JDK 1.3 實作 STD 演算法，Apriori 演算法，DHP 演算法，產生測試資料程式和前置處理程式。在 DHP 演算法的 hash 函數是以 h({x,y})=((order of x)*10+(order of y)) mode 7 的模式計算及產生 hash bucket 的數目與記錄 candidate count 值。

(3) 實驗資料庫:由產生測試資料程式隨機產生測試資料庫。其中物項集合，交易次數(frq)，資料筆數皆由亂數(random)產生，產生測試資料庫參數如表 9 所示:

表 9. 所有參數意義

D	原始資料交易記錄筆數
Ds	交易次數合併後資料交易記錄筆數
T	交易最長的物項(item)個數
I	平均可能高頻物項集合之物項個數
L	可能高頻物項集合之物項個數
N	資料庫所包含的物項(item)個數

我們用來實驗的資料庫有 6 個資料筆數(***D***)為 100K 的資料庫，各包含物項個數(***N***)為 500 個，可能高頻物項集合之物項個數(***L***)為 2500 個，交易最長的物項個數(***T***)有 6、8、10 個項

目，平均高頻物項集合之物項個數(***I***)為 3 或 4。六組測試資料庫如表 10 所示：

表 10.資料庫參數及代表意義

Database 縮寫	*N*	*T*	*I*	*D*
N500T6I3D100K	500	6	3	100000
N500T8I3D100K	500	8	3	100000
N500T10I3D100K	500	10	3	100000
N500T6I4D100K	500	6	4	100000
N500T8I4D100K	500	8	4	100000
N500T10I4D100K	500	10	4	100000

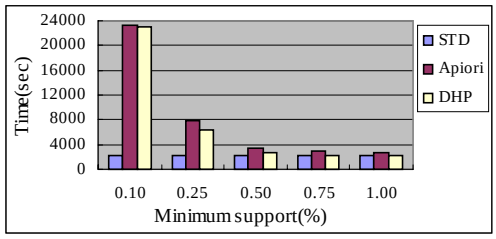
4.1 實驗設計

我們所提出的 STD 演算法是為一般消費行為的交易資料做資料掘取，因此分別設計產生不同物項集合的資料庫(D)，物項約有 500 個，資料庫筆數大約 100K，其每筆交易物項個數分別由 1 至 10，再經前置處理將資料庫(D)做交易物項依小至大排序後，相同交易物項集合做累加並記錄，最後依交易物項個數做由大至小的排序產生新資料庫(D_1)，而後 STD 演算法存取新資料庫(D_1)，Apriori 演算法、DHP 演算法存取原資料庫(D)，進而觀察各演算法的執行效能，並對其結果做一個評估比較。由這些觀察的結果，我們可以預期 STD 演算法在不同支持度的變化下比其他兩種演算法表現優異。以下我們針對 STD 演算法的效率與特性做了三種實驗。

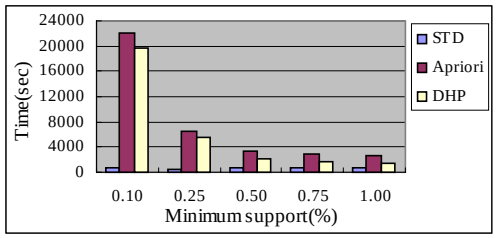
4.2 STD 演算法與 Apriori、DHP 演算法執行效率比較

我們在 D100K 資料筆數的資料庫，平均高頻物項集合為 3，交易物項集合的物項個數為 6、8、10，以 STD 與 Apriori、DHP 做比較，經多次的實驗產生圖 3 的實驗結果:

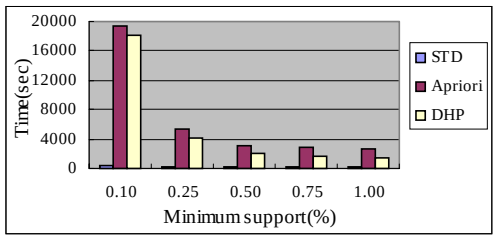
如圖 3 及表 11，我們可以看到實驗結果，在三個交易資料庫 N500T6I3D100K、N500T8I3D100K、N500T10I3D100K，最大交易資料物項個數為 6、8、10 中執行， STD 演算法在 0.1%到 1.0%支持度的變化中平均執行



(a) N500T10I3D100K



（b）N500T8I3D100K



(c) N500T6I3D100K

圖 3. STD、Apriori 與 DHP 在 D100K 之效能比較

表 11. STD、Apriori 與 DHP 在 D100K 效能比較表

支持度	Apriori/STD	DHP/STD
0.10%	22.4 倍	20.1 倍
0.25%	6.6 倍	5.3 倍
0.50%	3.3 倍	2.3 倍
0.75%	3.0 倍	1.8 倍
1.00%	2.6 倍	1.6 倍
平均	7.4 倍	6.7 倍

的速度比 Apriori 演算法快 7.4 倍與 DHP 演算法比較則快 6.7 倍，由此我們知道 STD 演算法在執行上確實比 Apriori、DHP 演算法優異。另外，STD 演算法在支持度自 0.1%到 1%的變化對於執行效率的變化差異不大(含前置處理時間)，但 Apriori 和 DHP 演算法執行，明顯可看出在支持度 0.1%非常耗費時間，主要是 Apriori、DHP 演算法產生大量的候選物項集合，大量的候選物項集合不僅佔用很大的主記憶體而且 I/O 頻率增加，並且產生的高頻物

項集合的物項個數增加，需要多次重複掃描資料庫，影響整體的執行效率。當支持度逐漸增大時，候選物項集合相對減少，需要主記憶體的空間也越小，I/O 頻率降低，產生的高頻物項集合的物項個數減少，重複掃描資料庫次數相對減少，執行效率也大幅的改善。對於 STD 演物項集合的物項個數的資料庫中非高頻物項集合的分解處理花費的時間大約相同，因此在支持度變化下受到的影響並不大，故執行的效率呈現穩定的狀態，但 STD 演算法受到最大的影響在於物項集合的物項個數，在於非高頻物項集合的分解過程中，將會花費較多的時間，物項個數越長分解次數多產生的子物項集合越多，假如交易物項集合的物項個數為 *l*，將做 *l*-2 分解處理並產生的子物項集合有(*l*！/ 2)個(不考慮相同物項集合來源的刪除)，我們在下一節探討物項集合的物項個數對 STD 演算法的影響。算法在支持度改變下，因先執行資料庫整理，將相同的交易物項集合的交易紀錄彙整成一筆資料並記錄交易發生次數而且依物項個數作排序，經此處理降低資料筆數，後再利用交易發生次數判斷是否為高頻物項集合，並在非高頻物項集合與交易物項集合的物項個數不相等時，做非高頻物項集合的分解，同時刪減同來源的子物項集合等處理，當交易資料執行到 1-物項集合時即產生所有最大的高頻物項集合，並停止處理，減少 1-物項集合的處理，在前述的處理下 STD 演算法比 Apriori 與 DHP 有較優異的表現，尤其在支持度低時會產生較長又較多的最大高頻物項集合時(例如實驗數據支持度 0.5%以下時)有更優異的表現。

4.3 交易物項個數的變化對 STD 演算法的影響

如圖 4 顯示，STD 演算法在不同交易物項集合的物項個數的資料庫，執行的效率有顯著的差異，我們知道 STD 演算法是從物項個數最長的交易物項集合進行，遇到不同物項個數

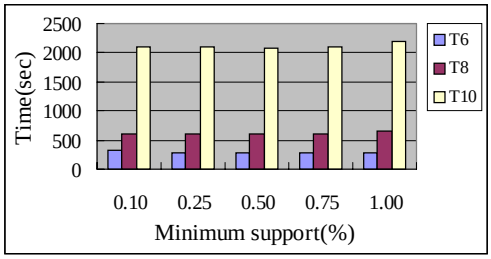


圖 4. STD 在 T6、T8、T10 的 I3D100K 資料庫之效能比較

的交易物項集合必須做分解處理，在分解的過程需要不斷的判斷是否存在已分解的子物項集合，並要做高頻與非高頻物項集合的判斷，因此分解程序是非常浪費時間。在此我們利用大約相同的資料筆數 100K，交易物項個數為最長為 6 個、8 個及 10 個的資料庫，進行 STD 演算法，在資料庫筆數沒有大幅改變下，只改變交易物項集合的物項個數，執行結果物項集合個數為 8 的執行效率比 6 個慢 2.13 倍，而物項集合個數為 10 的執行效率比 6 個慢 7.3 倍，其執行效率是最差。

我們知道在分解程序的過程中，在交易物項集合的物項個數 *l*，分解子物項集合必須做 *l*-2 次的分解程序，若在非高頻物項集合的物項個數為 *l* 做分解，將會產生最多為(*l*！/ 2)個子物項集合，且要不斷的判斷是否存在子物項集合及子物項集合是否為高頻物項集合，由此可知在此過程將會花費較長的時間，假設在物項個數 *l* 且有 *n* 筆交易物項集合皆非高頻物項集合，作分解至 2-物項集合，將產生 n × (*l*！/ 2)的子物項集合，若 *l* 的物項個數有 *n* 筆交易物項集合，*l*-1 的物項個數有 m 筆交易物項集合，2 的物項個數有 *k* 筆交易物項集合，若在分解過程中都沒有高頻物項集合出現，將會有 *n* × (*l*！/ 2) + *m* × ((*l*-1)！/ 2)+…….+ *k* × (3！/ 2) 的子物項集合個數出現 (3-物項集合分解至 2 物項集合後，2-物項集合不繼續作分解)，由上述公式我們知道影響執行效率最大的是資料筆數及交易物項集合的物項個數，而資料筆數的影響度沒有交易物項集合的物項

個數的大，資料筆數是以倍數，交易物項集合的物項個數是以階層的級數增加，當交易物項集合的物項個數越長對執行效率影響的程度是越大的。

由上述可知，在分解過程中若沒有發生高頻物項集合，在交易物項集合的物項個數為 10 時，將產生 10!/2(1.81×10^6)個非高頻物項集合且須做 8 次分解，因此執行的效率相對於 T6 產生 360 個非高頻物項集合做 4 次分解與 T8 產生 20160 個非高頻物項集合做 6 次分解較為緩慢。

4.4 支持度提高時的變化對 STD 演算法的影響

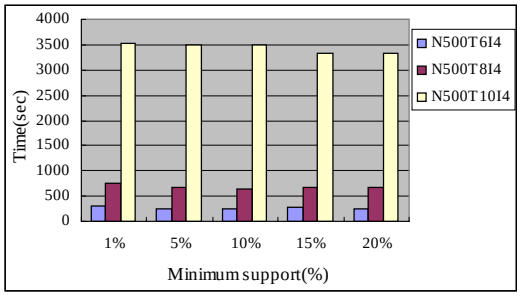


圖 5. STD 在不同支持度變化時效能比較

最後，我們針對當支持度大幅提高時，STD 演算法的執行效率變化做進一步的探討，看看是否有改變其特性。我們針對 100K 資料筆數的資料庫設定最大交易物項集合的物項個數分別為 6、8、10，並且提高某些物項的交易次數，產生三組 N500T6I4D100K、N500T8I4D100K、N500T10I4D100K 資料庫，此三組資料庫經 STD 演算法執行的結果，如圖 5 顯示，STD 演算法在支持度的提高下(1%至 20%)執行的的狀態，支持度的改變對執行的效率無顯著增加或減少，因為在物項個數分解產生子物項集合的過程都經過相同的步驟，故 6、8、10 個數的物項集合在不同的支持度下，其執行效率亦呈現穩定的狀態。

5. 結論

對於關聯式法則資料挖掘技術而言，最常以 Apriori 演算法為依循變化出各種不同的方法，改善主記憶體不夠、或 I/O 掃描過於頻繁等問題，以提昇執行的效率，在此基礎上做任何改善皆受到支持度的影響，當支持度小的時候，執行的效率較差，反之較好，造成時間無法控制。而我們的 STD 演算法跳脫 Apriori 演算法的架構，以新的思維重新建構邏輯架構來挖掘資料，以改善原缺失，減少主記憶體的浪費，避免重覆做 I/O 處理，使執行有非常好的效率並讓執行的時間不因支持度的變化，差異過大造成執行的時間無法掌握，此為本研究的重點。

本研究將資料庫在做資料挖掘前先經過前置處理，資料筆數已大幅減少，且依物項集合交易次數直接判斷是否為高頻物項集合，再由上而下快速產生高頻物項集合;在過程中大量減少記憶體的使用，刪除重複的高頻物項集合及非高頻物項集合，而且資料庫做一次的交易次數資料合併處理及一次的 I/O 掃描，執行的速度呈穩定的狀態並大幅提昇挖掘效率。STD 演算法在支持度小的時候執行的速度，非常迅速，以 Apriori 演算法為基礎的方法幾乎無法比擬，當支持度提升到某值以後，STD 演算法執行的速度幾乎沒有變化，維持穩定的狀態，且執行的速度也較 Apriori 演算法為基礎的方法為優越，這是本方法的一大特色。

參考文獻

[1] R. Agrawal, T. Imielinkski, and A. Swami, “Mining Association Rules Between Sets of Items in Large Database,” Proceedings of the ACM SIGMOD International Conference on Management of Data, pp.207-216, May 1993.

[2] R. Agrawal and R. Srikant, “Fast Algorithms

for Mining Association Rules in Large Database," Proceedings of the 20th International Conference on Very Large Data Bases, September 1994.

[3] Ming-Syan Chen, Jiawei Han, and Philip S. Yu, "Data Mining: An Overview from a Database Perspective," IEEE Transactions on Knowledge and Data Engineering, Vol. 8, No. 6, pp. 866-882, December 1996.

[4] Jong Soo Park, Ming-Syan Chen, and Philip S. Yu, "An Effective Hash Based Algorithm for Mining Association Rules", Proc. of ACM SIGMOD, pp. 175-185, May 23-25 1995.

[5] Jong Soo Park, Ming-Syan Chen, and Philip S. Yu, "Using a Hash-Based Method with Transaction Trimming and Database Scan Reduction for Mining Association Rules," IEEE Trans. On Knowledge and Data Engineering, Vol. 9, No. 5, pp. 813-825, October 1997.

[6] D. Lin and Z.M. Kedem, "Pincer-Search: A New Algorithm for Discovering the Maximum Frequent Set," Sixth Int'l Conf. on Extending Database Technology, March 1998.

[7] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth, "The KDD Process for Extracting Useful Knowledge from Volumes of Data," Communications of the ACM, Vol. 39, No. 11, pp. 27-34, November 1996.

[8] A. Savasere, E. Omiecinski, and S. Navathe, "An Efficient Algorithm for Mining Association Rules in Large Databases," Proc. of 21st VLDB, pp. 432-444, 1995.

[9] S. Brin, R. Motwai, J.D. Ullman, and S. Tsur, "Dynamic Itemset Counting and Implication Rules for Market Basket Data," 1997 ACM SIGMOD Conference on Management of Data, pp. 265-276, 1997.

[10] B. Dunkel and N. Soparkar, "Data Organization and Access for Efficient Data Mining," ICDE 1999, Sydney, Australia.

[11] M.J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li, "New Algorithms for Fast Discovery of Association Rules," American Association for Artificial Intelligence, 1997.

[12] M.J. Zaki, "Scalable Algorithms for Association Mining," IEEE Trans. on Knowledge and Data Engineering, Vol. 12, No. 3, pp.372-390, May/June 2000.

[13] Jiawei Han and Yongjian Fu, "Mining Multiple-Level Association Rules in Large Database," IEEE Trans. on Knowledge and Data Engineering, Vol. 11, No. 5, pp.798-805, September/October 1999.

[14] Jong Soo Park, Ming-Syan Chen, and Philip S. Yu, "Efficient Data Mining for Path Traversal Patterns," IEEE Trans. on Knowledge and Data Engineering, Vol. 10, No. 2, March/April 1998.

[15] Jiawei Han, S. Chee, and J. Y. Chiang, "Issues for On-Line Analytical Mining of Data Warehouses," Proc. of 1998 SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'98), Seattle, June 1998.