有效率的維護頻繁項目集之研究

顏秀珍 李御璽 郭育婷 顧家源
銘傳大學 銘傳大學 銘傳大學 銘傳大學
資訊工程學系 資訊工程學系 資訊工程學系 資訊工程學系
sjyen@mail.mcu.edu.tw leeys@mail.mcu.edu.tw teagirl07@gmail.com 496479451@ntnu.edu.tw

摘要

傳統的關聯規則探勘(Association Rule Mining)主要是從給定的交易資料庫中找出頻繁項目集。但在現實的應用中，交易行為是一直發生的，因此交易資料將會不斷地新增，而頻繁項目集也會隨著交易的新增而有所變動，使用者更是希望能夠以最短的時間得到最新的頻繁項目集。因此許多學者便開始研究，在資料不斷新增的情況下，能夠有效率地找出新的頻繁項目集，而這方面的研究也稱為漸進式頻繁項目集探勘或資料串流中的頻繁項目集探勘。近期來這方面的研究大多採用 Tree-based 架構，但是這些方法都存在著必須反覆的調整節點和花費大量記憶體空間的兩大缺點。因此，本研究提出一個有效率的演算法，且只需要存取頻繁項目集，當有資料新增時，只需調整少量的節點就可完成結構的更新。

關鍵詞：資料探勘、頻繁項目集、資料串流、交易資料庫

Abstract

Mining association rules is used to find out frequent itemsets in a given transaction database. However, new transactions will be continuously added into the transaction database, and thus the frequent itemsets will change with time in real applications. For users, they are eager for getting the latest frequent itemsets from the updated database as soon as possible in order to make the best decision. Therefore, it has become an important issue to work out efficient ways of finding the latest frequent itemsets when transactions keep being added to the database. This issue is also called “incremental mining of frequent itemsets” or “mining frequent itemsets in data streams.” In addition, “tree-based structure” has been recently adopted in most of the studies in this field, but they have shown the two common problems repeated adjustments of tree nodes and a need for large amounts of memory space. As a result, this paper proposes an efficient algorithm which only keeps frequent itemsets in the tree structure. When a set of new transactions is added, the tree structure can be updated by only adjusting few tree nodes for our algorithm.

Keywords：data mining, frequent itemsets, data stream, transaction database

1. 簡介

隨著電腦硬體的逐漸進步，電腦已經可以儲存大量資料，並可將資料進行計算、篩選、排序等等，找出有利於人類的資訊。而哪些資料是有利於人類的資訊呢?資料探勘(Data Mining)這名詞就在此產生出來；資料探勘是在大量資料中發掘出潛在未知可能有用的資訊，其中關聯規則[3]探勘(Association Rule Mining)就是資料探勘中一項重要的技術，最早是由 Agrawal 等人於 1993 年率先提出，其目的是從交易資料庫(Transaction Database)中找出常一起購買的商品組合，藉由顧客的消費行為，就可以知道大多消費者買了某商品後，很有可能也需要同時購買另一商品；此時，在

行銷上就可以使用促銷或推薦的方式來吸引顧客，達到更大的獲利。

以下為相關的名詞定義：給定一個交易資料庫，其中每筆交易包含交易編號(TID)與在該交易中所購買的商品，如表 1；若有相同商品出現於同一筆交易中，則在該交易中視為出現一次。令 $I=\{i_1,i_2,\ldots,i_n\}$為所有項目(Item)的集合，每一個項目可視為一種商品。集合 $X\subseteq I$，稱為項目集(Itemset)，用$\{a_1,a_2,\ldots,a_m\}$表示，其中 $a_m(1\le m\le n)$為一個項目，而項目集的長度為此項目集中所包含項目的個數。一個項目集的支持度(Support)為交易資料庫中包含此項目集的交易筆數與總交易筆數之比值。若此項目集的支持度大於或等於使用者自訂的最小支持度門檻值(Minimum Support Threshold)，則稱為頻繁項目集(Frequent Itemset)，否則稱為非頻繁項目集(Infrequent Itemset)。

表 1 交易資料庫

TID	Items
100	A,B,D
200	B,C
300	B,C,D,E
400	C

關於找出頻繁項目集，已經有多篇論文提出不同的演算法[4, 5, 6, 7, 11]，它們大多都是從給定的資料庫中找出頻繁項目集；但在現實生活中，交易資料及網頁瀏覽資料等，都是一直不斷增加的，而使用者更是希望能同步跟上這些即時的訊息，找出頻繁出現的項目做出有利的決策。因此如何在資料不斷新增的情況下，有效率地維持最新的頻繁項目集，便成為一個因應現實環境的研究議題。關於這方面的研究，最直觀的作法就是當使用者要求最新的頻繁項目集時，將原有的資料與新進的資料做整合之後，再重新進行探勘的動作。但為求良好的探勘效率，已有些研究[1, 2, 8, 9, 10, 12]朝向善用先前探勘結果，來提升資料新增時的探勘效率。現存演算法中，依結構大概可分為兩種，分別為 Apriori-like 及 Tree-based 演算法。由於 Apriori-like 演算法[9, 10]需要多次掃描資料庫，且花費很多時間產生候選項目集(Candidate Itemsets)與計算其出現在交易資料庫之次數，因此近期研究較少以 Apriori-like 的結構來進行探勘；而 Tree-based 演算法[1, 8]大多是將新增資料併入在先前建立用來儲存原始資料的樹狀(Tree)結構中，過程中因為要保有樹狀結構中頻繁項目集的資訊，所以必須將樹上的節點做新增、刪除或調整，這些改變樹節點的動作都是非常耗費時間的；因此有論文[12]提出將整個交易資料庫的資訊全部都建在樹上，當有資料新增時可以直接將所有資料併入樹上，不須做節點的刪除與調整，又不必考慮是否符合頻繁項目集的條件；但這樣的做法，變成是非常耗費儲存空間，而且在探勘的過程中，因為存在太多不需要的資訊，造成探勘速度變慢。基於以上兩種情況，本文提出一個只需要存取頻繁項目集，又不必做大量樹節點調整的方法，此法當新增資料進入時，會先判斷哪些項目須新建於樹上，哪些項目要從樹上刪除，將該刪除的節點從原始樹上移除後，把新增資料須被建立在樹上的節點，連同原始資料須新建於樹上的節點一起建入；如此一來，不但能省去調整節點的時間，也能夠只保留使用者需要的頻繁資訊。

2. 相關研究

在本章節中我們將介紹探勘頻繁項目集演算法與探勘漸進式頻繁項目集的相關研究。在 1994 年 Agrawal 等人提出 Apriori 演算法[4]，是利用一個頻繁項目集的子項目集必定也是頻繁項目集的這個原則，從交易資料庫找出所有頻繁項目集。此演算法掃描第一次資料庫產生長度為 1 的候選項目集(Candidate Itemsets)，所謂的候選項目集是指在交易資料庫中有可能成為頻繁項目集的項目集，再經由使用者自

訂的支持度門檻做篩選之後得到長度為 1 的頻繁項目集，接著用長度為 1 的頻繁項目集相互組合得到長度為 2 的候選項目集，經過第 2 次掃描資料庫和支持度的篩選後得到長度為 2 的頻繁項目集，如此反覆進行，利用長度為 k($k \geq 1$)的頻繁項目集組合成長度為 k+1 的候選項目集，再進行資料庫的掃描求出長度為 k+1 候選項目集的計數，之後經由支持度的篩選得到長度為 k+1 的頻繁項目集，直至無法組出候選項目集，演算法就此終止。根據上述 Apriori 演算法的描述，可歸納出兩大缺點:(1)需要多次掃描交易資料庫，若產生最大長度為 m 的候選項目集，就必須掃描交易資料庫 m 次，由於交易資料庫的資料筆數通常是很龐大的，因此多次掃描將會增加許多執行時間。(2)此演算法過程中產生許多候選項目集，且須花費時間去計數每個候選項目集出現在交易資料庫中的筆數，判斷是否有超過使用者自訂的支持度門檻，這種反覆產生候選項目集的方式，也影響了程式的執行效率。

在 Apriori 演算法提出後，出現許多以 Apriori 演算法為基礎並改善其效率的演算法，但由於還是使用 Apriori 演算法的架構，因此重複掃描交易資料庫與須產生候選項目集的缺點依然存在，直到 2000 年，Han 等人提出 FP-Growth 演算法[7]，此演算法以全新的架構來找頻繁項目集，同時也改善了上述 Apriori 演算法的兩大缺點。FP-Growth 演算法首先掃描交易資料庫第一次，找出長度為 1 的頻繁項目，再掃描第二次資料庫，將資料庫中的非頻繁項目刪除，並將長度為 1 的頻繁項目按照出現在交易資料庫中的次數由多至少排序，之後使用此排序、篩選過後的交易資料建成一個樹狀結構；由於樹狀結構上的項目都是頻繁項目，因此稱此樹狀結構為 Frequent Pattern Tree (FP Tree)。FP Tree 是由排序和篩選過後的交易資料一筆一筆的建到樹上，因此一條路徑(Path)或子路徑(Subpath)代表一筆交易，而路徑上的節點(Node)代表交易中的項目；若是交易中前面項目相同，則可共用路徑的節點達到資料壓縮，因此每個節點中須記載此項目在此路徑中出現的次數。另外為了方便在 FP Tree 中追蹤某個項目，相同項目的節點必須連接起來，因此產生 Header Table 來指向各項目第一個出現的節點，而這個連結稱為節點鏈結(Node Link)。在建構完成 FP Tree 之後，要開始從樹上找出所有頻繁項目集。首先由 FP Tree 上，項目出現次數最低的項目 j 開始，透過項目 j 的節點鏈結找出和項目 j 一起出現的路徑，為了避免找出重複的頻繁項目集，此演算法採用只取路徑的前序，也就是從項目 j 為起點，項目 j 以前的路徑才取出，而這些取出的路徑就形成項目 j 的條件式資料庫(Conditional Pattern Base)；完成後也同時找出條件式資料庫中各項目的出現次數，將符合支持度門檻的項目 x 加上項目 j，就可得到交易資料庫中所有包含項目 j 且長度為 2 的頻繁項目集。而長度為 2 的頻繁項目集{x,j}也會再建立自己的條件式資料庫，並找出屬於項目集{x,j}條件式資料庫中的頻繁項目，組成交易資料庫中所有包含項目 j 且長度為 3 的頻繁項目集；依此類推，直到 Header Table 中所有的頻繁項目都找過為止。FP-Growth 演算法為了快速建構出條件式資料庫，會將過程中每一個條件式資料庫建成條件式 FP Tree (Conditional FP Tree)，透過節點鏈結，可從前一個條件式 FP Tree 產生下一個條件式 FP Tree。FP-Growth 演算法主要的優點有下列 2 點:(1)相較於 Apriori-Like 演算法，FP-Growth 演算法只需要掃描交易資料庫 2 次。(2)FP-Growth 演算法不須產生大量候選項目集，也省下了計數候選項目集的時間。根據以上優點，FP-Growth 演算法大幅度改善了 Apriori-Like 演算法的執行效率；因此近期頻繁項目集探勘的演算法，大多以 FP-Growth 演算法的架構來加以改良。

接下來介紹漸進式頻繁項目集探勘，2003

年 Cheung 等人提出了 FELINE (FrEquent/Large patterns mINing with CATS TrEe)演算法[1]，此演算法將交易資料庫建成 CATS Tree(Compressed and Arranged Transaction Sequences Tree)，再使用類似 FP-Growth 演算法的探勘方式探勘 CATS Tree，找出頻繁項目集。此法首先掃描交易資料庫，將所有交易資料庫中的資料都建於 CATS Tree 上，而不須找出頻繁項目或將項目排序。當一筆交易要加入 CATS Tree 時，會由根節點開始，往它的子節點及子孫節點尋找可以跟之前路徑合併的節點，當找到節點合併後，則繼續往合併節點的子節點及子孫節點尋找下一個可以合併的節點，直到新進交易中找不到可以合併的節點為止;而剩餘沒有被合併的節點，則在最後被合併的節點之後建一條新的子分支。而在合併的過程中，若遇到子孫節點的出現次數大於祖先節點時，則需要將子孫節點移到祖先節點之前。建構完成 CATS Tree 後，使用者必須設定一個支持度門檻，才可探勘出頻繁項目集。當有新增資料進入時，只需將新增資料直接加入 CATS Tree 上，不需再重新建樹；更新後的 CATS Tree 再以類似 FP Growth 演算法的探勘方式找出新的頻繁項目集。FELINE 演算法雖然藉由交換節點的方式盡可能的壓縮樹狀結構，但資料壓縮的程度與交易進入的先後順序及交易內項目的排列順序皆有關，因此無法保證 CATS Tree 能有最大壓縮；而且 CATS Tree 所存放的是整個交易資料庫中的資料，因此需要更多的記憶體空間；除此之外，在尋找可以合併節點的步驟也需花費許多執行時間。

2005 年，Leung 等人提出 Can Tree(Canonical-order Tree)[12]，此法將整個交易資料庫的資料都儲存於 Can Tree 中，再以類似 FP-Growth 演算法的探勘方式來進行 Can Tree 的探勘。此演算法首先將交易資料庫中的每一筆交易項目做某個順序來排列，如字母順序，再將排序好的交易資料庫建構成樹狀結構，不需考慮頻繁項目或是出現在交易資料庫中的次數；由於每個交易內的項目都是依據某個標準來排列，在樹狀結構中，每條路徑也都是按照此排列順序，故稱此樹狀結構為 Canonical-order Tree。建構完成 Can Tree 之後，再以類似 FP Growth 演算法的探勘方式，找出頻繁項目集。當有資料新增時，只需掃描一次新增交易資料庫，將其加入 Can Tree；更新完成後，再重新以同樣方式進行探勘。由於每一筆資料都經過某順序進行排序，因此不需像 FELINE 演算法要花很多時間尋找可合併的節點；但此樹狀結構紀錄交易資料庫中的所有項目，而且節點共用的情況也不一定是最佳壓縮，因此記憶體的花費會非常龐大。

2008 年，Tzung-Pei Hong 等人提出 FUFP(Fast Updated FP-tree)演算法[2]，FUFP 演算法使用了 FP-Growth 演算法的架構，應用於資料新增的情況下，在資料新增時藉由更新樹上節點，達到不需重建樹的目的；而其探勘方法使用了與 FP-Growth 相同的探勘方式。此法首先使用了 FP Tree 的建樹方法，將交易資料庫中的頻繁項目建於樹上，當有資料新增時，須判斷以下 4 種情況：(1)在原始資料庫中為頻繁項目，且在新進資料庫中也是頻繁項目者；這樣的項目代表在更新過後的資料庫，必定也為頻繁項目，所以無須改變這些項目在樹上的節點。(2)在原始資料庫中為頻繁項目，但在新進資料庫中卻是非頻繁項目者；此時，要將原始資料庫符合這種情況的項目交易次數，與新增資料庫符合此情況的項目交易次數做加總，加總後的結果為更新資料庫的總出現次數，如此一來，就可判別哪些項目在新增資料進入後還是頻繁項目，哪些項目在新增資料進入後變成非頻繁項目，若有項目變成非頻繁項目，則須將此項目由樹上移除。(3)在原始交易資料庫中為非頻繁項目，但在新增交易資料庫中卻是頻繁項目；這種情況下，則須回原始資料

庫中搜尋符合此情形的項目，再將原始資料庫符合此情形的項目出現次數，與新增資料庫符合此情形的項目出現次數做加總，得到更新資料庫的總出現次數，再判斷哪些項目由原本非頻繁項目變成頻繁項目，找到此項目之後，須將其項目建到樹上。(4)在原始資料庫中為非頻繁項目，而在新增資料庫也為非頻繁項目者；這種情況將不予考慮，因為此情況在更新後必定也為非頻繁項目。最後，再將更新完成的 FUFP Tree 以 FP-Growth 演算法的探勘方式進行探勘，得到頻繁項目集。根據以上的描述知道了此演算法在一開始建樹時跟 FP Tree 一樣須掃描 2 次資料庫，而且在判斷原始交易資料庫中為非頻繁項目，新增交易資料庫中為頻繁項目的過程中，必須回原始交易資料庫搜尋，造成執行效率降低。

3. 研究方法

我們的研究方法採用了 FP Split[11]的架構來進行建樹。首先掃描一次原始交易資料庫 DB，將每一個項目出現交易編號記錄下來，如圖 1，因此可以很容易知道每一個項目出現在交易資料庫中的次數。假設使用者自訂的支持度門檻為 50%，則可以知道至少出現在此交易資料庫 2 次以上，才會被稱為是頻繁項目；接著我們將符合頻繁項目的項目，做出現在交易資料庫中的次數由高而低排序，得到頻繁項目出現的次數為 B=3，C=3，D=2，因此排序為 BCD，再由排序較前面的項目先進行建樹。

DB

TID	Items
1	B,A,D
2	B,C
3	D,B,E,C
4	C

→

Items	TID
A	1
B	1,2,3
C	2,3,4
D	1,3
E	3

圖 1 儲存交易編號

一開始將項目 B 建於樹上，由於樹上還沒有任何節點，因此不需做任何判斷；樹上的每個節點皆包含項目、出現在此路徑上的次數與此項目出現在此路徑上的交易編號，如圖 2。當有第二個項目要建於樹上時，則須做以下 2 個步驟：(1)判斷是否需要往遇到節點的子節點發展，使用新進項目與樹上遇到節點的交易編號做交集來判斷；由例子知道接下來新進的項目為 C，出現的交易編號為{2,3,4}，因此將項目 C 的交易編號與在樹上遇到節點 B 的交易編號做交集，交集出來的結果為{2,3}，代表項目 B 與項目 C 同時有出現在{2,3}這兩筆交易中，因此項目 C 中交易編號為{2,3}的這兩筆交易可以往節點 B 的子節點發展，如圖 3。(2)判斷是否需要往樹上遇到節點的兄弟節點發展，此判斷需要使用新進項目與樹上遇到節點的交易編號做差集。延續範例，在項目 C 做完判斷是否要往樹上遇到節點的子節點發展後，我們要將新進項目 C 的交易編號{2,3,4}再與樹上遇到節點 B 的交易編號{1,2,3}做差集，得到的交易編號結果為{4}，此結果代表交易編號為{4}的這一筆交易有出現項目 C，但沒有出現項目 B，因此項目 C 中交易編號為{4}的這一筆交易需要往節點 B 的兄弟節點發展，如圖 4。之後進入的項目，每當遇到樹上節點皆根據以上 2 種判別方式來進行建樹。因此當項目 C 在樹上建構完成後，接下來要建入項目 D；當項目 D 一開始要建入樹上時，首先會遇到樹上的第一條路徑的第一個節點 B，將項目 D 的交易編號與節點 B 的交易編號分別做交集與差集，交集後的結果為{1,3}，代表交易編號為{1,3}的這兩筆交易有同時出現項目 D 與項目 B，因此可以往節點 B 這條路徑的子節點發展，向下發展則可遇到節點 C；而差集後的結果為空集合，則代表有出現項目 B 的交易編號包含所有出現項目 D 的交易編號，因此無須往節點 B 的兄弟節點發展。判斷完節點 B 之後，我們得知項目 D 的{1,3}筆

交易資料會往節點 B 的子節點 C 發展，跟上述一樣用交集與差集的兩個方式來判斷節點 C，項目 D 的交易編號{1,3}與節點 C 的交易編號{2,3}做交集之後，結果為交易編號{3}，代表交易編號為{3}的這一筆資料，同時有出現項目 B、C 和 D，所以此筆資料可往節點 C 的子節點發展，且由於節點 C 已無子節點，因此將交易編號為{3}的項目 D 建於節點 C 的子節點，如圖 5；在做完交集之後，繼續將項目 D 的交易編號{1,3}與節點 C 的交易編號{2,3}做差集，差集後得到的交易編號為{1}，則代表交易編號為{1}的這一筆交易中，有出現項目 D，但是沒有出現項目 C，所以要將交易編號為{1}的項目 D 建於節點 C 的兄弟節點，如圖 6。由於已無頻繁項目須建於樹上，因此原始交易資料庫的樹就建構完成，在 FP Split 建構完成後，就可以 FP-Growth 的探勘方式找出頻繁項目集。

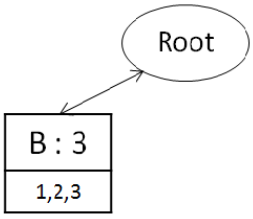

圖 2 建立節點

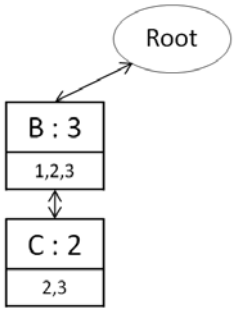

圖 3 建立子節點

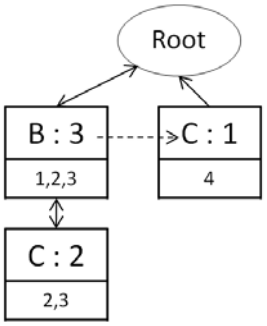

圖 4 建立兄弟節點

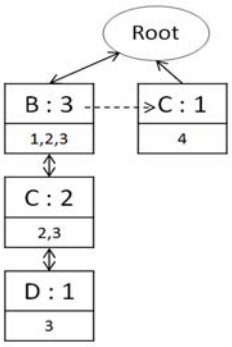

圖 5 建立子節點 D

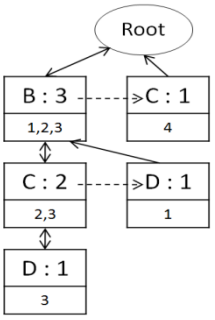

圖 6 建立兄弟節點 D

當有新增資料庫進入時，我們將修改原始資料庫已建好的樹，而不重新建樹。表 2 為新增交易資料庫 db，掃描一次新增交易資料庫得到每個項目出現的交易編號，如表 3；此時須計算新增資料庫的門檻值，由於使用者自訂的支持度門檻為 50%，所以在此新增資料庫中出現 1 次以上的項目，就稱為新增資料庫的頻繁項目，因此 A、B、C、E 為新增交易資料庫頻繁的項目。之後在更新原始資料庫所建的樹過程中，會遭遇以下 4 種情況：(1)在原始資料庫中為頻繁項目，且在新進資料庫中也是頻繁項目者；(2)在原始資料庫中為頻繁項目，

但在新增資料庫為非頻繁項目者；(3)在原始資料庫中為非頻繁項目，但在新進資料庫中為頻繁項目者；(4)在原始資料庫中為非頻繁項目，且在新進資料庫中也是非頻繁項目者。其中的第 4 種情況，在原始資料庫中不是頻繁項目，因此不會出現於樹上，且在新增資料庫中也不是頻繁項目，所以完全不會影響到原始樹上的節點，當然也就不需考慮此種情況之情形。首先考慮第一種情形，如例子所示，原始交易資料庫為頻繁項目的項目有 B、C 和 D，而新增交易資料庫中為頻繁項目的項目有 A、B、C 和 E，所以我們得知符合第一種情況的項目有 B 和 C；之後將項目 B 與項目 C 的交易次數分別做加總，也就是將原始交易資料庫與新增交易資料庫中相同項目的出現次數相加，因此得到項目 B 在總交易資料庫中的出現次數為 4 次，項目 C 在總交易資料庫中的出現次數也為 4 次，其中總交易資料庫代表原始交易資料庫與新增交易資料庫的合併；由於原始交易資料庫為頻繁項目的項目已建於樹上，因此之後新增交易資料庫為頻繁項目且與原始交易資料庫中頻繁項目一樣的項目，只需改變樹上節點的交易次數與交易編號即可。判斷完第一種情況之後，再繼續探討第二種情況，由範例得知，原始交易資料庫中為頻繁項目的項目有 B、C 和 D，而新增交易資料庫中為非頻繁項目的項目有 D，所以我們得知項目 D 符合以上所述的第二種情況。此時必須至新增交易資料庫儲存交易編號的地方，也就是表 3，找出項目 D，並察看此項目在新增交易資料庫中所出現的次數為何；在此例中，新增交易資料庫儲存交易編號的地方未出現項目 D，代表項目 D 完全沒有出現在新增交易資料庫中，因此項目 D 在總交易資料庫中出現的次數為 2 次。由於目前是計算總交易資料庫中的次數，所以必須知道總交易資料庫的門檻值，經由計算後得知總交易資料庫中的項目要出現 3 次以上的項目，才會被稱為總交易資料庫中的頻繁項目；因此項目 D 在總交易資料庫中不符合頻繁項目的條件，需要由樹上刪除所有節點 D，且由於節點 D 均是葉節點，所以無需做判斷直接刪除即可，如圖 7；但若要刪除的節點非葉節點，則須判斷此刪除節點的子節點與兄弟節點是否有相同的項目，如果有則須將兩相同項目的節點合併，如果沒有則直接將刪除節點的子節點與父節點相連。

表 2 新增交易資料庫

	TID	Items
db	5	B,A
	6	A,E,C

表 3 儲存交易編號

Items	TID
A	5,6
B	5
C	6
E	6

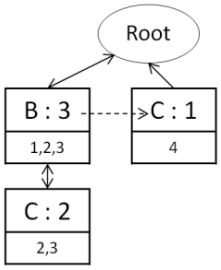

圖 7 刪除範例的節點 D

在刪除樹上節點後，接下來要判斷第三種情況，判斷在原始交易資料庫不是頻繁項目集，但在新增交易資料庫中為頻繁項目的項目，由此例可得知為項目 A 與項目 E。此時須至原始交易資料庫儲存交易編號的地方，也就是圖 1，找出項目 A 與項目 E，並計算這兩項目在總交易資料庫中出現的次數；計算後得知在總交易資料庫中項目 A 出現 3 次，項目 E 出現 2 次，

所以只有項目 A 從原本不是頻繁項目變為頻繁項目，因此項目 A 必須加到樹上。在此之前，由於我們只將原始交易資料庫所建的樹做節點刪除的動作，還未將新增交易資料庫中的任何項目建到此樹中，因此接下來我們將同時處理須新增在樹上的項目 A 與新增交易資料庫中符合總交易資料庫門檻值的項目。首先，將新增資料庫中符合樹上頻繁節點的項目由新增交易資料庫儲存交易編號的地方(表 3)找出，因此找出項目 B 與項目 C，分別出現的交易編號為{5}與{6}，接著將這兩個項目依原始樹上節點的排列順序依序建於樹上，一開始將項目 B 建入，首先遇到樹上第一條路徑的第一個節點 B，由於項目相同，因此直接將新進的項目 B 加於此節點 B 中，如圖 8；其中新加入的交易編號將另外儲存，直至新增交易資料庫中符合樹上頻繁節點的項目都建構完成，才會將交易編號做合併，如此一來新進項目的交易編號就不需與原始交易資料庫的編號做運算。接下來要建入樹上的是新增交易資料庫中的項目 C，一開始遇到第一條路徑的第一個節點 B，此時將項目 C 的交易編號{6}與節點 B 中屬於新增資料的交易編號分別做交集與差集，交集後的結果為空集合，代表項目 C 無須往節點 B 的子節點發展；做完交集後再做差集，結果為{6}，則代表項目 C 中交易編號為{6}的這一筆交易須要往節點的 B 的兄弟節點發展，所以遇到了樹上的節點 C，且由於新進項目 C 與節點 C 為同一項目，因此直接將新進的項目 C 加於此節點 C 中，如圖 9；當新增交易資料庫中符合樹上頻繁節點的項目都建構完成後，則將樹上節點的交易編號合併，如圖 10。

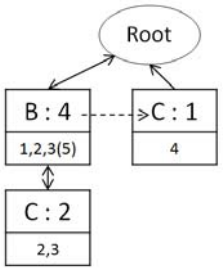

圖 8 新增資料庫的節點加入

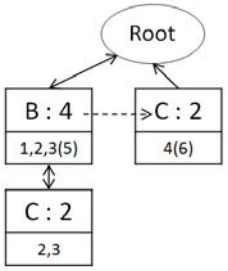

圖 9 新增資料庫節點加入

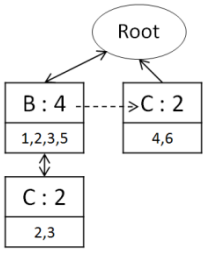

圖 10 合併交易編號

之後只剩下項目 A 還未建立於樹上，由於項目 A 不管是在原始交易資料庫還是新增交易資料庫都是還沒建到樹上的狀態，因此先將項目 A 出現在原始交易資料庫中的交易編號與出現在新增交易資料庫中的交易編號做聯集，得到項目 A 在總交易資料庫中的交易編號為{1,5,6}，接著就使用和原始交易資料庫的建樹方式一樣將項目 A 建於樹中，如圖 11。建構完成後，此樹也更新完成，再利用 FP-Growth 演算法的探勘方式，探勘出現新的頻繁項目集為 B、C、A。

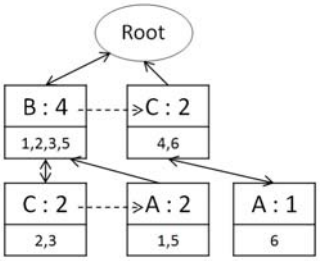

圖 11 節點加入

4. 討論

在過去關於漸進式頻繁項目集探勘或在資料串流的環境下探勘頻繁項目集的研究，大多存在以下幾種缺點：(1)需要存取交易資料庫中的所有項目與出現的次數，使記憶體的花費非常龐大。(2)節點共用的情況不一定是最佳壓縮。(3)由於樹狀結構不一定是最佳壓縮，因此樹有可能會長的非常龐大，造成探勘效率不佳。(4)關於這方面的研究，大多使用 FP Tree 的架構，因此也和 FP Tree 的建構方式一樣需要掃描 2 次交易資料庫建樹，而且在原始資料庫中為非頻繁項目，新增資料庫為頻繁項目的情況下，也必須回原始資料庫掃描，影響執行效率。(5)有些此方面的研究為了使樹狀結構保持最佳壓縮的狀態，因此反覆調整樹上的節點，但也影響了探勘的效率。基於以上缺點，本研究提出一個改善的方法，此法只需存取頻繁項目的資訊；由於本研究使用了 FP Split 的架構來建樹，所以只需掃描 1 次交易資料庫，且在新增資料庫進入時，也無須回原始資料庫掃描；另外，在資料新增時，我們採取直接新增和刪除樹上節點的方式，省去反覆調整樹節點的時間。因此我們認為本研究提出的方法可提升尋找頻繁項目集之效率。

參考文獻

[1] William Cheung and Osmar R. Zaïane, “Incremental Mining of Frequent Patterns Without Candidate Generation or Support Constraint,” ***in Proceedings of the 7th International Database Engineering and Applications Symposium (IDEAS),*** pp. 111-116, 2003.

[2] Tzung-Pei Hong, Chun-Wei Lin and Yu-Lung Wu, “Incrementally fast updated frequent pattern trees,” ***Expert Systems with Applications : An International Journal Vol. 34, No. 4,*** pp. 2424-2435, 2008.

[3] R. Agrawal, T. Imielinski and A. Swami, “Mining Association Rules between Sets of Items in Large Databases,” ***in Proceedings of the 12th ACM International Conference on Management of Data (SIGMOD),*** pp. 207-216, 1993.

[4] R. Agrawal and R.Srikant, “Fast Algorithms for Mining Association Rules in Large Databases,” ***in Proceedings of the 20th International Conference on Very Large Data Bases (VLDB),*** pp. 487-499, 1994.

[5] Mohammad El-Hajj and Osmar R. Za¨ıane, “COFI-tree Mining: A New Approach to Pattern Growth with Reduced Candidacy Generation,” ***Workshop on Frequent Itemset Mining Implementations (FIMI’03) in conjunction with IEEE-ICDM.***

[6] Ke Wang, Liu Tang, Jiawei Han and Junqiang Liu, “Top Down FP-Growth for Association Rule Mining,” ***in Proceedings of the 6th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining,*** pp. 334-340, 2002.

[7] Jiawei Han, Jian Pei and Yiwen Yin, “Mining Frequent Patterns without Candidate Generation,” ***in Proceedings of***

the 2000 ACM International Conference on Management of Data (SIGMOD), pp. 1-12, 2000.

[8] Jia-Ling Koh and Shui-Feng Shieh, "An Efficient Approach for Maintaining Association Rules Based on Adjusting FP-Tree Structures," ***in Proceedings of the 12th International Conference on Database Systems for Advanced Applications (DASFAA),*** pp. 417-424, 2004.

[9] David W. Cheung, S.D. Lee and Benjamin Kao, "A General Incremental Technique for Maintaining Discovered Association Rules," ***in Proceedings of the 5th International Conference on Database Systems for Advanced Applications (DASFAA),*** pp. 185-194, 1997.

[10] David W. Cheung, Jiawei Han, Vincent T. Ng and C.Y. Wong, "Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique," ***in Proceedings of the 12th International Conference on Data Engineering (ICDE),*** pp. 106-114, 1996.

[11] Chin-Feng Lee and Tsung-Hsien Shen, "AN FP-SPLIT METHOD FOR FAST ASSOCIATION RULES MINING," ***in Proceedings of the 3rd International Conference on Information Technology : Research and Education (ITRE),*** pp. 459-463, 2005.

[12] Carson Kai-Sang Leung, Quamrul I. Khan and Tariqul Hoque, "CanTree: A Tree Structure for Efficient Incremental Mining of Frequent Patterns," ***in Proceedings of the 5th IEEE International Conference on Data Mining (ICDM),*** pp. 274-281, 2005.