在少樣商品或短交易長度情況下挖掘關聯規則

陳家仁 陳彥良 陳禹辰
中央大學資訊管理學系 中央大學資訊管理學系 東吳大學企業管理學系

摘要

從交易資料庫中挖掘出的關聯規則可以幫助組織實行目標行銷，如進行市場區隔、選擇目標顧客、改進賣場陳設或組合搭售商品。以往有關的研究大多假設在單一商店的商品項目即可能達到數萬種以上，同時顧客可能會同時採購非常多樣化的商品。但在實際的世界中，許多商店如專賣店、精品店、速食店、餐廳、保險公司、百貨公司中的專櫃等等，所販售的商品種類可能只有數十至數百種不到；此外，一般消費者在多數的商店中每次購買的商品的種類通常也不會太多。基於上述兩種情況，本文發展一個全新的挖掘關聯規則作法，針對挖掘關聯規則時最耗時的步驟加以改進，在掃瞄資料庫一次後，將資料庫的內容儲存於一個樹狀結構中，再利用此樹狀結構產生關聯規則。如此將可大幅減少 I/O 的時間，讓使用者能更快產生關聯規則，並且不需在掃瞄資料庫前即指定 minimum support，可以動態給定 minimum support 而不用重新掃瞄資料庫。

關鍵字：資料挖掘、關聯規則、交易資料庫

Mining Association Rules When Number of Items is Limited or Transaction Length is Short

Jia-Ren Chen, Yen-Liang Chen
Department of Information Management, National Central University
Yu-Chen Chen
Department of Business Administration, Soochow University

ABSTRACT

The problem of mining association rules is to find the associations between items in a large database of sales transactions. Basically, the past researches studied the problem with the assumptions that a great number of different items are sold in a store and a customer may buy quite a few items in a single round of purchase. No doubt, such situations fit in with the retailing store or convenience store well. However, there are many situations in practice that only a limited number of items are sold or the average transaction length is short. The possible examples include shopping in luxury goods stores, electric appliance stores, musical instrument stores, cigar stores, wine stores, glasses stores, watch stores, make up stores, underwear stores and so on. In view of this difference, this paper develops a new algorithm for mining association rules in such a special situation: small transaction length and hundreds of different items. Our experiments show that the developed algorithm outperforms the currently best algorithm, FP tree algorithm, designed for mining association rules in general situations.

Keywords : Data mining, Association rule, Transaction database

壹、 緒論

爲了提升品質，改善工作效率以及增強本身的競爭力，許多企業與組織的交易處理均已電腦化。這意謂著有數以百萬計的交易資料庫同時正在被使用，其中每一個資料庫中均可能儲存著大量的交易資料。如何能自動化的從大量的資料中找出有用的資訊與知識，增加交易資料的價值以支援決策所需，就變的更有其需要。因此資料挖掘（ data mining ）逐漸變成一個重要的研究領域，而『資料挖掘就是從資料庫中發現知識，將隱含的、先前並不知道的，和潛在有用的資訊從資料庫中萃取出來的過程(Han and Kamber 2001)。』

一筆交易記錄通常包含有交易日期及交易的項目，另外若配合上其他的交易機制，如會員資格、信用卡等，還可以納入購買人的資料。資料挖掘技術可以協助決策者進行所謂的「菜籃分析(basket analysis)」，從交易資料中找出關聯規則，例如：80％購買牛奶的顧客同時會購買麵包。利用這些規則可以幫助組織實行目標行銷，進行市場區隔、選擇目標顧客、分析顧客行爲、改進賣場陳設、組合搭售商品、發掘不良率等等。

以往有關關聯規則的研究都假設：「交易資料庫是在單一商店的商品項目可能達到數萬種以上的環境中產生的」，例如超級市場所販賣的商品種類會達到五、六千種，量販店更高達一、二萬種；此外，顧客在此類型的商店中，常常一次就購買數十種不同的商品。基於以上的假設，在尋找關聯規則的過程中，目前多數的作法都必須先找出所有超過 minimum support 的 large item set ，才能依據 large item set 來產生關聯規則。這個執行步驟必須多次的掃瞄資料庫，花費許多 I/O 的時間，佔去整個作業流程絕大部分的時間，往往成爲決定整個資料挖掘作業執行效率的關鍵。一般而言，由於系統 I/O 是最耗費時間的動作，掃瞄資料庫的次數愈多，演算法的執行效率會愈不佳。

在實際的世界中，並不是每一家商店都會有成千上萬的商品。在日常生活中隨處可見的許多商店，所販售的商品種類可能只有數十至數百種不到，例如各種專賣店、服飾店、精品店、速食店、餐廳、保險公司、百貨公司中的專櫃等等均是如此。此外，通常只有在販賣一般民生消費用品的商店中，例如量販店、超市、個人商店中，顧客才比較有可能會經常性地一次就購買十幾種或者更多不同種類的商品。在其他衆多非販售民生消費用品的商店中，顧客每次選購的商品種類通常只有個位數，例如在電子產品、家電、家具、書籍、鐘錶、一些高單價商品等等商店中，一次購買十多種商品的交易是不常發生的。

本研究是以上述兩種常見的情況爲基礎，發展一個全新的挖掘關聯規則作法，針對挖掘關聯規則時最耗時的步驟加以改進，在掃瞄資料庫一次後，將資料庫的內容儲存於一個樹狀結構中，再利用此樹狀結構產生關聯規則。如此將可大幅減少 I/O 的時間，讓使用者能更快產生關聯規則，並且不需在掃瞄資料庫前即指定 minimum support ，可以動態給定 minimum support 而不用重新掃瞄資料庫。

貳、文獻探討

有關挖掘關聯規則 (association rule) 的研究，是目前最重要的資料挖掘問題之一。它的目的是要從銷售的交易資料庫中，發現項目 (item) 間的關聯。若在許多交易中，我們發現某些項目的出現會引發其他項目的出現，這樣的關聯關係，即可以用關聯規則的型式加以表達。例如：牛

奶 麵包。

在探討關聯規則的挖掘之前，我們必須先了解 minimum support 和 minimum confidence 的概念。minimum support 界定一個規則必須涵蓋的最少資料數目，minimum confidence 則界定這個規則的預測強度。規則的 support 和 confidence 可以評估規則是否有趣，當挖掘演算法所找出的規則滿足使用者訂定的最小 support 和 confidence 的門檻時，這個規則才算成立（ Agrawal and Srikant 1994 ； Han, Pei and Yin 2000 ）。

關聯規則有許多種類，大體上可以將它分成三類（ Han and Kamber 2001 ）：

1. 以屬性值的型態為基礎：

如果我們所關注的只是 item 是否出現，這種便稱爲布林值的關聯規則 (Boolean association rule)，例如「牛奶 ⇒麵包 (support=2%,confidence=60%)」即屬於這類關聯規則。如果我們也一併關注 item 的購買單位數，這種便稱爲有重複項目的關聯規則（ association rule with repeated items)（陳彥良等 2001 ；沈清正等 2002 ），例如「 2 單位牛奶⇒3 單位麵包 (support=2%,confidence=60%)」即屬於這類關聯規則。如果我們所要描述的規則其項目或屬性是一個數值，這種就稱爲數量關聯規則 (quantitative association rule)。但因爲數量關聯規則的可能性太多，所以我們必須把數量值切割成不同的區間[1]，才有辦法產生關聯規則。如下面的例子， X 是代表消費者的一個變數。

年齡 (X,"40...45")^ 收入 (X,"7 萬 ...8 萬)⇒購買 (X, 海外基金)

2. 以規則中所涵蓋的資料維度為基礎：

如果在關聯規則中的項目或屬性僅參照單一的維度時，我們稱之爲單一維度關聯規則(single dimensional association rule)，例如我們將「牛奶⇒麵包」的關聯規則寫成「購買 (X," 牛奶 "⇒購買 (X," 麵包 "))，則其著眼的是「購買」這個維度。反之，如果關聯規則中的項目或屬性參照兩個以上維度時，便稱爲複合維度關聯規則 (multidimensional association rule)，例如上述定量的關聯規則中的例子，便包含了「年齡」、「收入」以及「購買」等三個維度。

3. 以規則中所涵蓋的抽象層級為基礎：

如果在關聯規則中的項目或屬性可以屬於不同的概念層級，例如「年齡 (X," 中年 ")⇒購買 (X," 味全果汁牛奶 ")」(" 中年 " 對於年齡而言屬於較高層級概念，但 " 味全果汁牛奶 " 對於購買項目而言屬於較低層級概念），則稱這類規則爲跨層級關聯規則 (multilevel association rule)。反之，如果沒有參照到不同層級的項目或屬性規則，則稱爲單一層級關聯規則 (single-level association rule)。

目前挖掘關聯規則的演算法可依需不需產生 candidate itemset 的作法分爲兩類，例如 Frequent-Pattern tree 與 Apriroi-like approach。此兩者最主要的差異在於， FP-tree 並不產生 candidate itemsets（ Han, Pei and Yin 2000 ），它將資料庫壓縮在 Frequent -Pattern tree 的結構中，避免多次的高成本的資料庫掃瞄；後者是需要產生 candidate itemset 的方

[1] 可以事先就切好，或根據資料分布情況來切割，或根據語意、模糊函數、資訊含量等不同方式切割。

法，這一類的研究相當多（ Agrawal and Srikant 1994 ； Li et al. 1999 ； Park et al. 1997 ； Pasquire et al. 1999 ； Savasere et al. 1995 ； Toivonen 1996 ），大體上皆針對 Apriroi algorithm (1994) 的作法進行改進，所以統稱爲 Apriroi-like approach 。

Apriroi algorithm 主要有兩個階段，第一階段是找出所有超過 minimum support 的項目集合（即 large itemset ），第二階段再從 large itemset 找出關聯規則。他的精神是 frequent itemsets 的子集合也必定是 frequent itemsets ，利用 frequent (k-1)-itemsets 來產生 frequent k-itemsets ，在每一個回合都需掃瞄資料庫一次，用以計算 candidate itemsets 的次數。 Apriroi algorithm 的瓶頸在於會產生大量的 candidate itemsets ，例如有個長度爲一的 frequent 1-itemsets 將會在下一個回合產生超過個長度爲二的 candidate itemsets 。每一回合 candidate itemsets 產生後還需要掃瞄資料庫來比對，它佔了整個作業絕大部分時間，因此 Apriroi-like approach 的研究，均將焦點放在如何有效率的找出 large itemset 來，這方面過去所用的改進方法有使用 hashing 技術（ Park et al. 1997 ）、使用 sampling 方法（ Toivonen 1996 ）、使用 Partition 資料庫的技術（ Savasere et al. 1995 ）、使用 closed itemset lattice （ Pasquire et al. 1999 ）、使用 item clique(Li et al. 1999) 。

Frequent-Pattern tree 是不產生 candidate itemsets 作法的代表（ Han, Pei and Yin 2000 ）。因爲不用產生 candidate itemsets ，所以只需掃瞄資料庫兩次，可以避免多次的高成本的資料庫掃瞄，節省了大量 I/O 的時間，因此整體的效率相當不錯。在目前現有的研究報告中，還沒有其他的挖掘方法在效率上能比 Frequent-Pattern tree 方法快，而且已運用在實際的產品 DB-Miner 中。 Frequent-Pattern tree 的作法可分爲兩個階段，首先是建立 Frequent-Pattern tree ，接著是 mining Frequent-Pattern tree ，主要步驟如下：

建立 Frequent-Pattern tree ：

1. 第一次掃瞄資料庫，找出符合 minimum support 的 frequent 1-itemset ，即只有單一個 item 的 pattern 。
2. 將每一筆紀錄中 frequent items 依出現在資料庫中次數的多寡，由大至小排列。
3. 第二次掃瞄資料庫並建立 Frequent-Pattern tree 。

Mining Frequent-Pattern tree ：

1. 針對 FP-tree 中的每一個 node 建立其 conditional pattern base 。
2. 接著對每一個 conditional pattern base 分別建立其 conditional FP-tree 。
3. 再對 conditional FP-tree 進行挖掘，並逐次增加包含在 conditional FP-tree 的 Frequent Pattern 。
4. 如果 conditional FP-tree 中有包含一條路徑，就可容易地列舉出所有 pattern 。

參、問題定義

如何從交易資料中挖掘出關聯規則，可以分解成兩個子問題：第一個問題是如何從交易資料中找所有滿足 minimum support 的 itemsets ，也就是要找出所有大項目集合（ large itemset ）；第二個問題是如何利用大項目集合去產生關聯規則。

先前有關挖掘關聯規則與如何產生 large itemsets 的研究，對交易資料庫的組成（如每筆交易紀錄的長度、資料庫包

含的項目個數）並無特別考量或限制，主要是針對演算法的延展性（ scalability ）加以考量，希望能對所有交易資料庫提供一個一致的解決方案。但是如前所述，一旦我們回歸到實際運用的層面，便會發現在眞實的世界中，存在有各式各樣不同的交易資料庫，除了販賣上萬種商品的量販店、超市所使用的大型交易資料庫之外，更常見的是各式各樣的專賣店、電子電器商店所使用的中小型交易資料庫。

不同類型的商店所使用的交易資料庫當然也會有差異。除了交易的內容不同外，最主要的差異在於組成資料庫的 item 個數不同，原因在於商店所販賣商品的種類不同；另一個差異是交易記錄的平均長度不同，因爲消費者在不同商店的消費習慣不同，而且每家商店的商品平均單價不同，平均單價高每次所購買的數量自然較少。因此本文將對交易資料庫的組成做兩種限制：

1. 交易資料庫包含的項目個數，限制在一般專賣店所販售的商品種類個數以內。
2. 交易紀錄之平均長度，交易記錄的長度是指在同一筆交易中，所購買商品的種類，例如有一筆交易的內容爲甲顧客購買了二個 A 、一個 B 及四個 C 等三種商品，則此筆交易的長度爲三。

在這兩類的限制下，本文將提出一新的演算法，只需掃瞄交易資料庫一次即可產生所有的 large itemsets 。

肆、演算法

一、建立All-subset tree

假設一筆交易記錄是由一組 item 所組成，例如範例資料庫中 TID 100 是由 {A C D} 三個 item 組成，則 TID 100 所有的子集合爲 {A,C,D,AC,AD,CD,ACD}，每個子集合可以視爲一條條不同的路徑（ path ），依序將路徑加入在一樹狀結構中，此樹狀結構每一個節點內容有 item 和此 item 出現的次數，父節點所代表的 item 必須比子節點大，在加入一條新的路徑時只有將最末端的節點次數加一，我們以圖 1 說明新增 TID 100 到樹狀結構的過程。圖 2 說明依序新增交易資料 TID 200 = {B C D} 和 TID 300 = {A B D} 到樹狀結構的完整過程。

例如要加入 {A C D} 這一個子集合時，首先將 A 、 C 、 D 這三個 item 由小至大排序，之後可以得到要新增的路徑爲 A→C→D ，然後由根節點（ root ）出發，檢查根節點的第一層子節點中有無路徑的起始點 A 這個 item ，若不存在則新增一子節點 A ，若已存在有 A 這個 item 則重複先前的動作，檢查 A 節點的子節點中有無路徑的第二個點 C 這個 item ，若不存在則新增節點 C ，若已存在則檢查子節點有無路徑的下一個節點 D ，重複以上動作直到檢查終端的節點 D 是否已存在，並將新增路徑的終端節點 D 的 support 加一。

本文建立此樹狀結構的作法是先找出每一筆交易記錄所有可能的子集合，再將此筆交易記錄的所有子集合依序加入樹狀結構中，所以此樹狀結構的所有路徑就是資料庫中所有交易記錄的所有可能子集合，可以用來代表整個資料庫，本文稱爲 All-subset tree 。由於建立 All-subset tree 是將交易記錄的所有子集合依序加入並加以紀錄，所以資料庫中任一種組合一定可以在 All-subset tree 中找到，也就是說在挖掘關聯規則時所需要的 candidate itemsets 都已被包含在 All-subset tree 中。在 All-subset tree 中只要是從根節點爲起點的任一條路徑（不包含根節點），都代表著交易資料庫中某一筆交易記錄的

子集合，而路徑末端節點中的次數是代表此子集合在資料庫中出現的次數，例如圖2中B→D這條路徑就代表{BD}這個子集合，而節點D的次數爲2，代表{BD}在Database D中出現過二次。

比較圖2中完整的All-subset tree與圖3 Database D的子集合統計，由Database D所建立的All-subset tree長度等於一的路徑有A(2)、B(2)、C(2)、D(3)，與由一個item所組成的子集合相等，長度等於二的路徑有A→B(1)、A→C(1)、A→D(2)、B→C(1)、B→D(2)、C→D(2)，與由二個item所組成的子集合相等，長度等於三的路徑有A→B→D(1)、A→C→D(1)、B→C→D(1)，與由三個item所組成的子集合相等，而且三者的support也相同。所以建立All-subset tree後就可以產生所有的candidate itemset及其support，在All-subset tree中一條路徑就代表一個candidate itemset。

觀察All-subset tree後可以歸納出以下特徵：

1. 路徑由N個節點所組成就代表N個item的子集合，其中不包含根節點。
2. 交易資料最大的長度就是All-subset tree的高度。
3. All-subset tree第一層的節點代表組成交易資料庫的所有item。
4. All-subset tree由第N層爲末端節點所組成的所有路徑，代表出現在資料庫中所有由N個item所組成的集合，例如第一、二層的節點所形成的路徑就是出現在資料庫中所有兩個item的組合。

和Apriroi algorithm相比較，若將minimum support設爲0，則Apriroi algorithm第一回合掃瞄資料庫就是要找出All-subset tree第一層的節點，第二回合是要找出第二層的節點，其餘依此類推，而本文所提的演算法只需一次的掃瞄即可。

在minimum support不爲0的情況下，只要將All-subset tree中次數不符的節點刪除即可，剩下的節點就是large itemsets，就可以用來產生關聯規則。

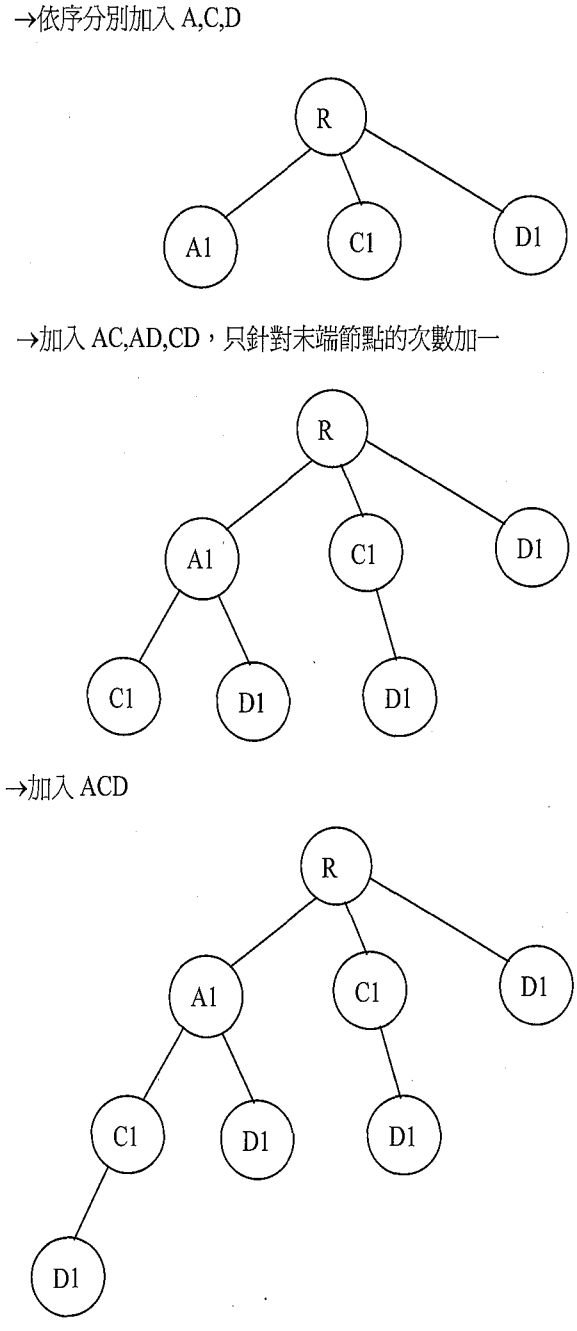
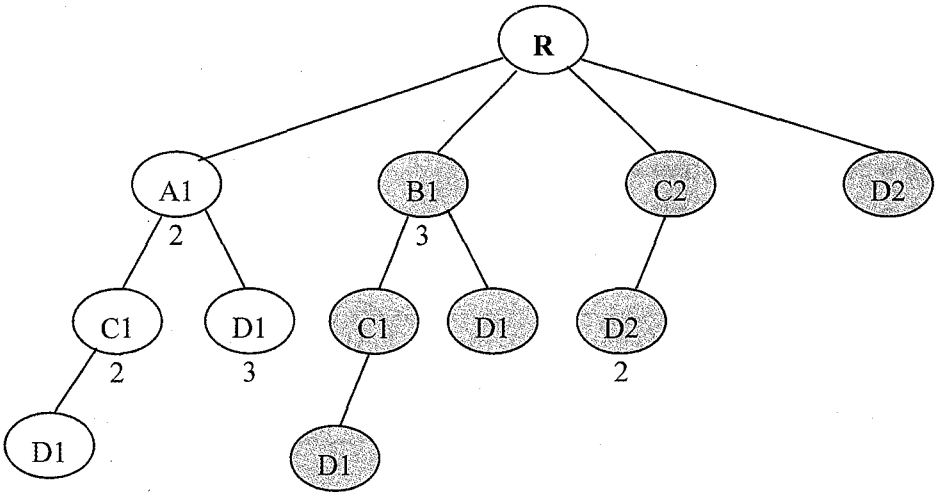


圖1：新增第一筆交易記錄

→在 ACD 中再加入 B C D



→加入 ABD

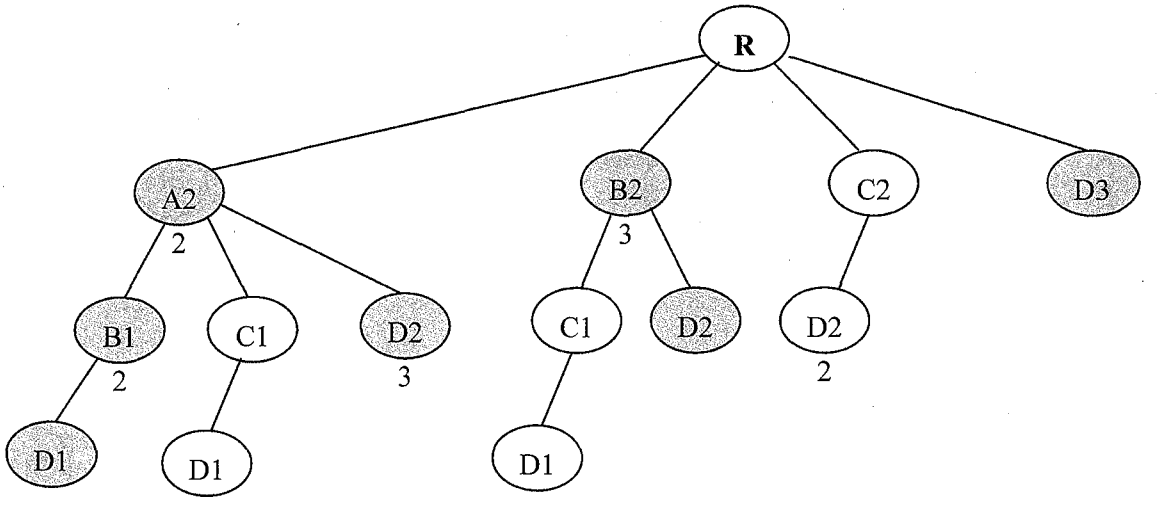


圖2：建立All-subset tree

Item 個數	子集合	出現次數
一	A	2
	B	2
	C	2
	D	3
二	AB	1
	AC	1
	AD	2
	BC	1
	BD	2
	CD	2
三	ABD	1
	ACD	1
	BCD	1

圖3：Database D的子集合統計

演算法如下：

```
Algorithm Construct All-subset tree
Input : All records T in transaction database DB

Output: All-subset tree

Method: Call construct All-subset tree( root , T )

 Procedure    Construct All-subset tree(node n, set T)
//*n 是目前的節點， T 是一筆記錄中所有的 item
//*n.item 是節點 n 所儲存的 item
{
     if (n=nil)
     {       //* 此時樹還不存在
             node root=new node
             root.item=null
             Construct All-subset tree (root, T)
      }
      else
      {
           //*subset.size 代表 subset 所含的元素個數
          //*n.addchild(child) 新增節點 child 爲節點 n 的兒子
          subset = all items that greater than n.item
          if(subset is a null set)
               return;
```

```
            else{
                for(j=0; j<subset.size; j++)
                {
                    if n has a child node where item=s[j] then
                        //* 若 item 已經存在則次數增加
                        increase the support of this child
                    else
                        node child=new node
                        child.item=subset[i]
                        n.addchild(child)
                    end if
                    Construct All-subset tree (child, subset)
                }
        }
    }
```

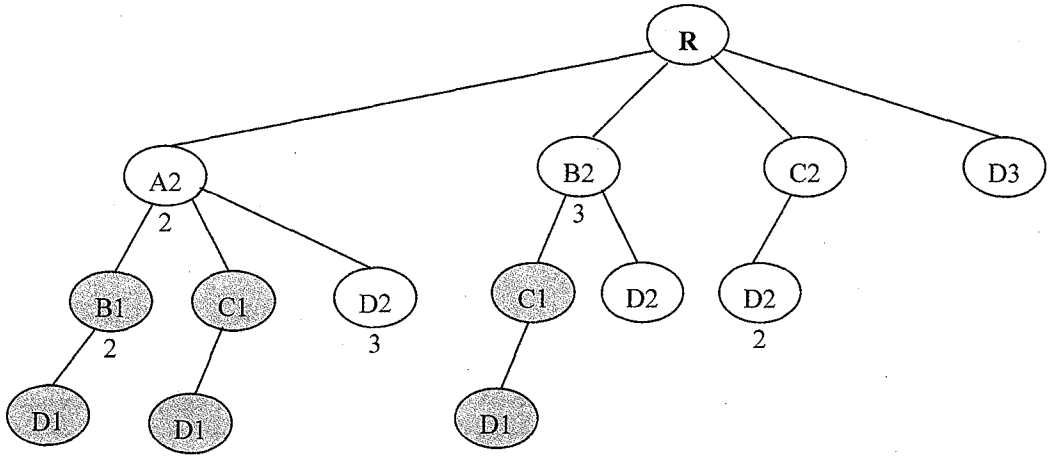
二、找出large itemsets

在建立 All-subset tree 後接著是要產生 large itemsets。首先需將 All-subset tree 中未達 minimum support 的節點刪除，然後將所有的路徑印出即可。圖 4 是從 Database D 所建立的 All-subset tree 中找出 large itemsets，其中 minimum support 設爲二。

演算法如下：

```
Algorithm Generate large itemsets
Input : All-subset tree and minimum support
Output: large itemsets
Method: Call Generate large itemsets ( root , minimum support ) Procedure Generate
large itemsets (root, minimum support)
{
    traversal the All-subset tree
    {
    //*node.support 代表節點 node 的 support 值
    if  node.support < minimum support then delete the node
    }
    print all paths in the All-subset tree
}
```

→設 minimum support 爲二，將未滿 minimum support 的節點刪除



→過濾後的 All-subset tree

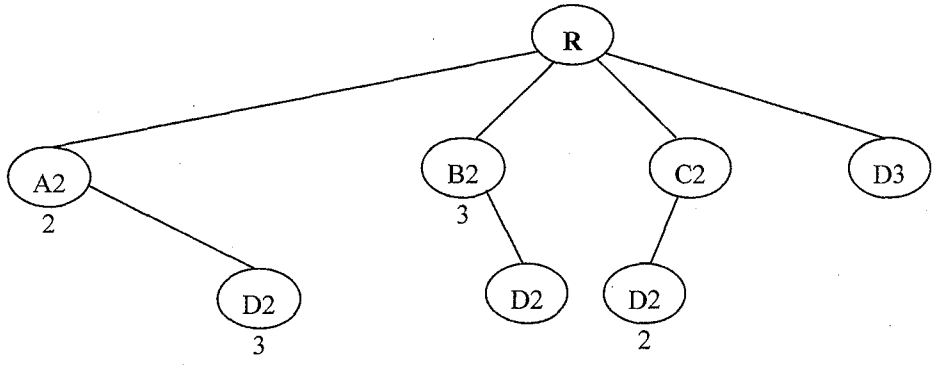


圖4：從All-subset tree中找出large itemsets

伍、實驗模擬

一、實驗設計

本研究進行一系列的實驗以評估前述演算法的效率。實驗的環境爲 windows 2000，硬體爲 Intel Pentium III-933 CPU 及配備 1024 MB RAM，此外程式是以 Delphi 5 撰寫。

實驗用的測試資料使用模擬眞實銷售環境的模擬交易資料，本文是以 [3] 文中所使用的方式產生，此方法爲多數挖掘關聯規則的研究所援用。在模擬資料中我們定義了以下的參數：

D：交易資料之總筆數。

T：每筆交易紀錄之平均長度。

N：交易資料庫中 item 的個數。

I：潛在 large itemset 的平均長度

L：潛在 large itemset 的上限個數

在眞實的交易記錄中，每筆交易的長度會群集在一個平均數附近，只有少部分的交易會出現大量的 item，而 large item set 也是有相同的狀況。在產生交易記錄時先依照一個μ等於 T的 poisson distribution 產生每筆交易的長度，若是每一個

item 被購買的機率 p 都相同，則假設交易紀錄之長度會服從 binomial distribution，所以交易紀錄之長度的分佈會趨近於一個 μ 等於 N × p 的 poisson distribution。

接著要決定每筆交易記錄的內容，我們由一群潛在的 large itemsets T 中選出一組加入，若選出的 itemset 無法因爲長度的關係無法完全放入交易記錄中，則將無法加入的部分 itemset 加入到下一筆記錄中。潛在的 large itemsets T 的個數等於 L，且 itemset 長度服從一 μ 等於 I 的 poisson distribution，由於 large itemset 中常會出現相同的 item，所以記錄中部分的 item 是由先前的 itemset 中挑選出部分出來。

另外，若未有其他特別說明則 minimum support 皆固定設爲 1 %，L 設爲 2000。

二、結果分析

1. FP-tree 與 All-subset tree 二者比較

本次實驗中資料庫的參數爲 N=500，D=100K。由圖 6 可得知 All-subset

名稱	T	N	I	D	資料量（MB）
T3I2D100K	3	500	2	100K	1.6
T4I4D100K	4	500	4	100K	2.0
T5I4D100K	5	500	4	100K	2.5
T6I4D100K	6	500	4	100K	3.2
T5I4D10K	5	500	4	10K	0.3
T5I4D50K	5	500	4	50K	1.4
T5I4D80K	5	500	4	80K	2.2
T5I4D150K	5	500	4	150K	4.1

圖5：模擬交易資料庫參數設定

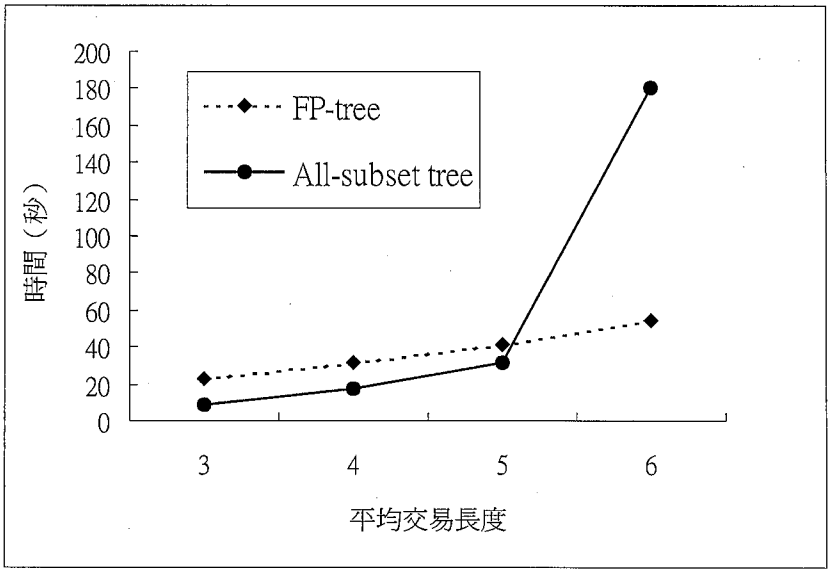


圖6：FP-tree與All-subset tree不同交易長度之比較

tree 在平均交易長度五以內的效率非常良好，但超過長度五以後，所需要的時間大幅增加，原因在於 All-subset tree 在建構時完全沒有過濾的動作，所以當交易長度愈長，就代表交易紀錄的子集合越多，建構 All-subset tree 所需節點增多，所佔記憶體的空間也越多，而且新增路徑時所需搜尋的空間也隨之增加，當逼近記憶體的容量上限時，執行效率就明顯變差。

一筆長度為 n 的交易記錄，就有 (2^n-1) 個子集合，每一個子集合都需要一個節點來表示，所以可以得知，交易記錄的長度對 All-subset tree 的大小有顯著的影響。依據本研究模擬實驗的經驗，在主記憶體為 1024MB 的環境中， All-subset tree 的節點數目在接近九百萬個時，效率就會明顯變差。也就是在平均交易長度五到六之間，在平均交易長度等於五時效率還仍佳，但到平均交易長度六時因為節點數已經超過九百萬，所以造成時間大幅增加。

而 FP-tree 因為有使用 minimum support 來過濾 large 1 的 itemset ，所以時間增加的幅度並不像 All-subset tree 那麼激烈，但由於需掃瞄資料庫二次，所需 I/O 的時間較多，所以在平均交易長度較短時效率較差。

2. 不同 item 個數

本次實驗中資料庫的參數為 T=5 ，D=100K 。由於 item 個數增加，在固定的交易長度下，每一個 item 出現的次數會減少，因此滿足 minimum support 的 large item 個數也會減少，所以 FP-tree 所需的時間會緩慢減少，所減少的時間是節省於建構 FP-tree 的 large 1 itemsets 減少。

圖 7 顯示在 item 個數較少時， All-subset tree 的執行效率會比 FP-tree 來的佳，而隨著 item 個數的增加， All-subset tree 的時間曲線是以平緩的幅度增加，時間增加的原因也是因為 All-subset tree 的節點增加的影響。

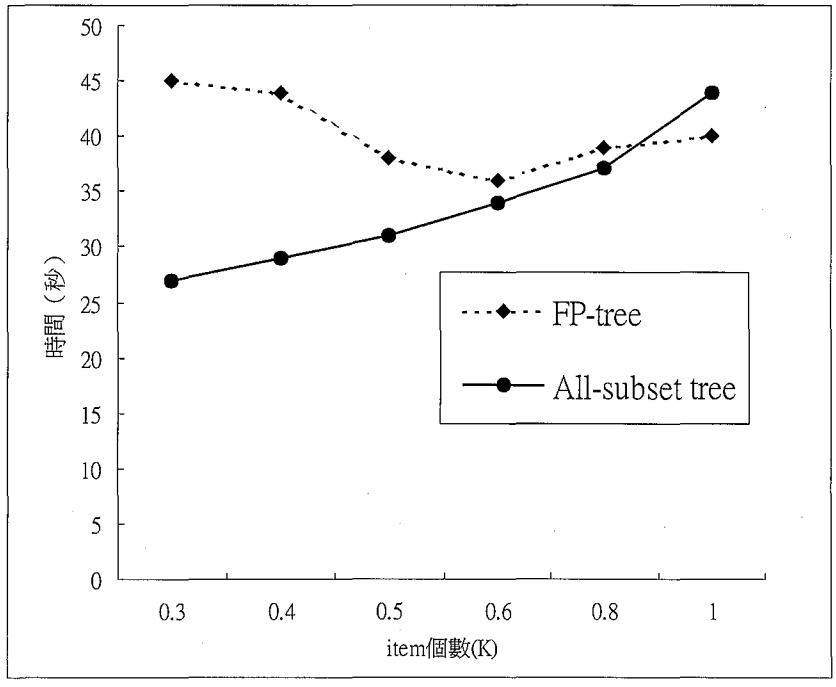


圖7：FP-tree與All-subset tree不同item個數之比較

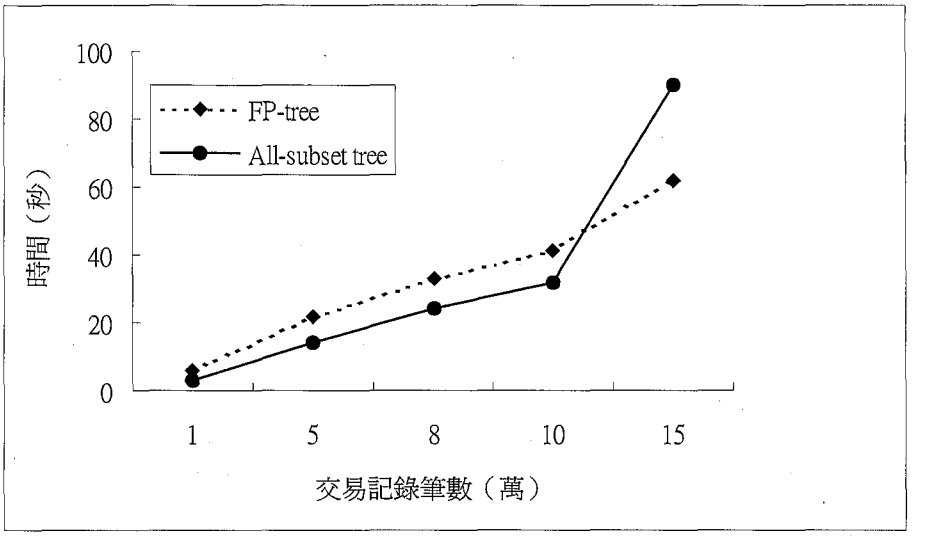


圖8：FP-tree與All-subset tree不同交易筆數之比較（T=5）

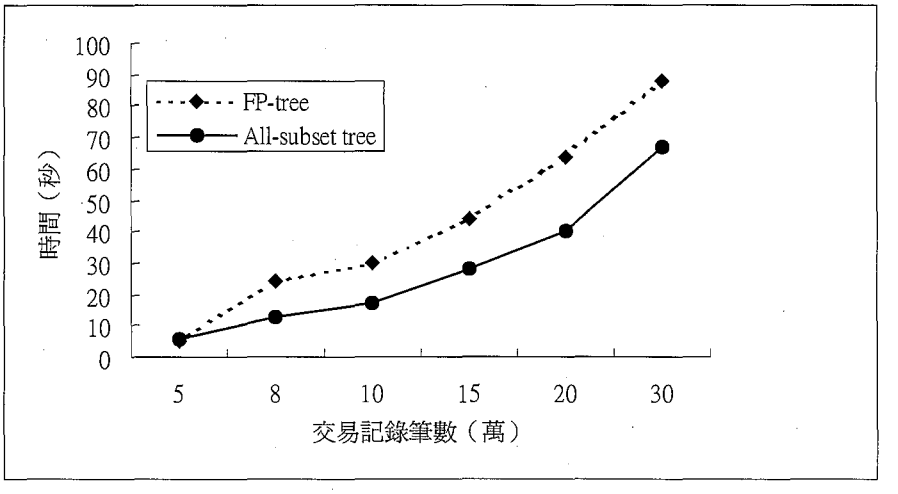


圖9：FP-tree與All-subset tree不同交易筆數之比較（T= 4 ）

3. 不同的交易紀錄筆數

本次實驗中資料庫的參數爲 T=5 及 4，N=500。交易筆數的增加，對於 All-subset tree 演算法來說是，交易記錄子集合組合會更多，因爲會有更多的交易讓不同的 item 有組合在一起的機會，也就使得 All-subset tree 的節點增加。另外由圖 9 可以發現當 T=4 時 All-subset tree 演算法的效率都較 FP tree 爲佳。

4. 不同的 minimum support

由於在建構 All-subset tree 時並不需設定 minimum support，所以不同的 minimum support 對於 All-subset tree 所需時間並無影響，但對於 FP-tree 就有明顯的影響，若給予較小的 minimum support，則在建構 FP-tree 及 FP-growth 時都需更長的時間，在圖中可以明顯的得知越上方的曲線其 minimum support 越小。

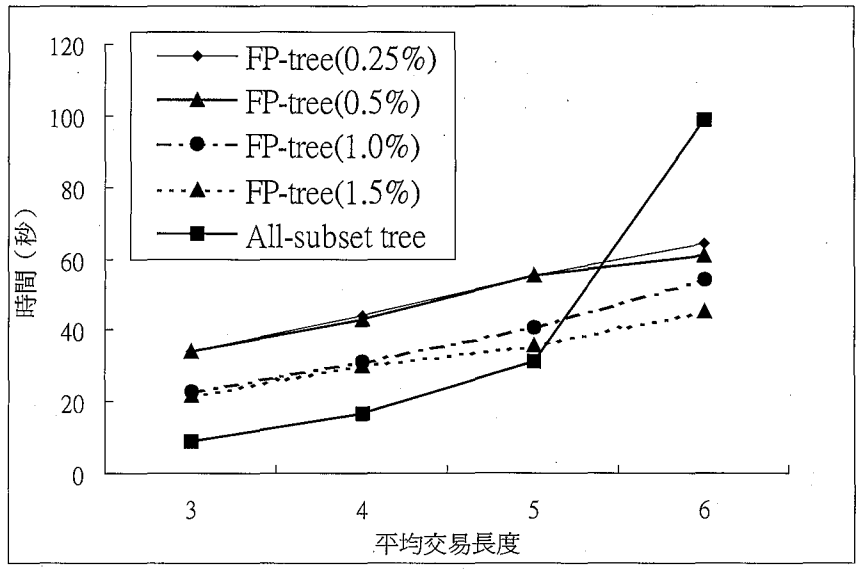


圖10：All-subset tree與不同的minimum support的FP-tree之比較

三、實驗結論

由以上實驗得知，在本文所提的資料庫限制條件：即「item 個數較少，交易長度較短的資料庫」下，All-subset tree 演算法的執行效率皆非常良好，其最主要的原因在於只需掃瞄資料庫一次，而且並無其他額外的過濾動作，只要單純的建立 All-subset tree 即可。

而執行時間會大幅增加來自於三方面，一是交易長度增加，二是 item 個數增加，三是交易筆數增加，這三個原因都會使 All-subset tree 的節點增加，其中以交易長度增加影響最大。依據本研究的經驗，在先前所提出的軟、硬體環境下，當 All-subset tree 的節點在九百萬個以內，也就是交易資料庫有九百萬種以下子集合，演算法的執行效率都令人相當滿意，一旦慢慢超越此一門檻值，則效率會急速下降，因此在實際使用此演算法時，必須對這三方面加以考慮。

另外有一點不可忽略的是，傳統的資料挖掘中，我們往往必須嘗試各種不同 minimum support 門檻值，但 All-subset tree 演算法卻是不用事先給定 minimum support 的。特別是當交易長度短時，support 值會偏低，因此若要找出足夠的規則，就必須降低 minimum support 門檻；不然找出的規則將都只是少數熱門商品的組合。此時決策者必須不斷嘗試不同的 minimum support 門檻，以使規則不至於太少而門檻值又不會太低，因前者會使某些重要的訊息被遺漏，相反的，後者會造成雜訊太多的困擾。這時，若我們是使用傳統的演算法便需要不斷重新掃瞄資料庫，重做完整的資料挖掘動作。而 All-subset tree 演算法則因爲將整個資料庫壓縮在記憶體中，除非資料有更動，否則 minimum support 的變動不會影響到 tree 本身的結構，反映出來的只是會產生不同的關聯規則，完全不用再掃瞄資料庫。關於這項優點所能節省的時間是在本實驗中所無法反映出來的。

四、相關應用

從模擬實驗的結果可知，在特定範圍的交易資料庫中，本文所提的演算法執行效率較佳，而此範圍涵蓋了各種專賣店、

服飾店、精品店、速食店、餐廳、保險公司、百貨公司中的專櫃等等商店所使用的交易資料庫。

在其他可能領域的應用方面，資料個人性的研究也十分適合使用本研究的演算法。在以資料個人化爲基礎的資料挖礦中，不論是 item 個數、交易紀錄筆數都會比以單一商店爲基礎的交易資料庫來的小，但因爲顧客人數一定是遠大於商店數目，因此演算法所需執行的次數會高出許多。儘管每次執行時所減少的時間不多，但是若再乘上顧客人數，則在效率上很可能就會有較大的差距。

另外在日益受重視的 WWW 資料挖礦方面，觀察幾個知名的網站（除入口網站外），如 Ebay 、 Amazon 、 New York Times 、 Wall Street journal 等等，可以發現通常使用者在五個 click 之內即可找到想要的資訊，就是完成一次交易，若將一個網頁視爲一個 item ，也可以解讀爲一筆的交易所包含的 item 個數是五以內。所以 WWW log 的資料挖掘亦適合使用本文之演算法。

五、如何應用於一般交易資料庫

本文所提出的方法適合用於少樣商品或短交易長度，而某些實際交易資料庫（特別是大賣場）其所包含的商品可能多達上萬種而平均交易長度也可能多達數十個商品項目，於此種情形下，若透過一些適當的修正，本文的方法仍能應用於此種資料庫的挖掘。底下談到幾種可能的作法。

第一種可能的做法是在演算法中應用抽樣的概念以做適當的前過濾。即當 all-subset tree 演算法掃描資料庫若干筆後（例如 5000 筆資料），我們立刻就進行樹的剪除動作，把一些 support 值低於某一門檻值的節點通通刪除掉，如此可以得到一棵修剪後的樹 T'。接著我們繼續掃描資料庫尚未掃描的其他交易，針對每一交易，我們會檢視看有哪些 T' 中的節點合乎此一交易，並把這些節點的 support 值加一（注意，於此階段我們不加入新的節點）。在完成掃描後， T' 中所有節點的 support 值都是正確的。唯一可能發生的錯誤是如果有某些商品在前面很少出現（建樹的時候），但到了後面卻很常出現，則這些 frequent itemsets 會被遺漏。理論上這當然可能發生，但如果我們隨機抽取建樹的資料夠多時，事實上其機率相當低。此種做法仍然只需要一次資料庫的掃描，而且又可以大幅降低記憶體的需求量，所以效率上應該可以更好，但缺點是不能保證結果一定正確。

第二種可能的做法是我們把演算法改成兩階段的資料庫掃描，第一階段的掃描當中，我們可以計數每一商品項目的出現次數，然後我們只保留前 K 名（例如前 400 名，因爲圖 7 顯示如果商品數目不到 400 時，我們的方法遠比 FP-tree 好）的商品或是達到 minimum support threshold 的商品，到了第二階段，我們便可以只針對這些商品建立 all-subset tree 。此種做法需要增加一次資料庫的掃描時間，但同時因爲第一階段會過濾掉許多的商品，所以第二階段建立 all-subset tree 的時間會變短，而且記憶體的需求量也會減少。

第三種可能的做法是在應用本文的方法前先以一些限制條件對資料加以篩選。而此種先篩後挖的做法，事實上和一般資料挖掘實務的做法是相同的，因爲關聯規則最爲人所詬病的，就是在不做任何篩選過濾的情況下會產生非常多的規則，讓使用者淹沒在一大堆的規則中，使用者根本不知道要如何來解讀和利用這些規則。因此，一般的資料挖掘系統都會要求使用者事先指定資料的範圍與條件限制，資料必須先合乎限制，才能進行挖掘。而經過篩

選後，往往只會剩下某些類別、廠商、時期、售價、銷量的商品，其商品種類不會太多，因此很適合應用本文的方法去挖掘。

第四種做法是把本文的方法應用在資料倉儲環境下的資料挖掘，因爲資料倉儲系統中往往會對資料建立多層級之概念階層，例如所有的商品可以區分爲不同之大分類，大分類下可再分爲不同的中分類，再往下爲小分類，最後則爲個別商品編號。而在資料倉儲中，使用者可以把資料往上匯聚（ roll up ），由詳細的資料變爲概括式的資料，或者使用者也可以把資料往下分解（ drill down ），把一筆概括式資料還原成多筆的細部資料。當我們在資料倉儲環境挖掘關聯規則時，我們所挖掘的規則便不只是挖掘出在單一概念層級中的關聯規則，而是會在不同的資料階層中，找出多個概念層級的關聯規則，例如：不僅挖掘出『８０％的顧客購買“麵包”，則他也會購買“牛奶”』的關聯規則，也可以挖掘出『８０％的顧客購買“小麥麵包”，則他也會購買“巧克力牛奶”』的較低層概念的關聯規則。因此，當我們把資料往上匯聚時，商品項目會變少而且每一交易的長度也變短了，此時就非常適合使用本文的演算法去挖掘規則。

陸、結論與建議

本文首先指出兩個在交易資料庫中常出現的現象，一是資料庫的 item 個數並不都是數以萬計，許多資料庫 item 數只有數百種而已；另一個現象是大多數的交易長度都不會太長，在個位數內都是合理的。針對這兩種常見的情形，本文提出了 All-subset tree 的演算法，能在這兩項前提下快速的產生關聯規，並能變動產生關聯規則所需 minimum support ，而不需重新建構 All-subset tree 。

此外，本文的演算法也證明了，如果對資料庫的組成做一些限制，想在只掃瞄資料庫一次的情況下，即可找出關聯規則是可行的。除了零售業的交易資料庫，資料挖掘的技術在各個領域裡也急速的發展，其他領域、行業的交易資料庫 item 個數或交易紀錄長度可能更短，更符合容易本文所給予的限制，也可以運用於其中。

經由本文的實驗模擬可得知， All-subset tree 的演算法在執行的效率上仍存有瓶頸，要擴大 All-subset tree 所能適用的資料庫，或成爲一通用的演算法，可以從二方面來探討：

1. 可發展平行處理的方法：因爲只需掃瞄資料庫一次，所以並沒有先後次序的因果關係，此點有利於發展平行處理的方法，增進執行的效率。
2. 發展過濾的方法：可以有效減少 All-subset tree 在記憶體所佔的空間，能擴大適用的資料庫範圍。

此外，還有一些可能的後續研究，包括如果已知原資料庫的 all-subset tree ，當有新的交易發生，如何設計一個遞增式（ Incremental ）演算法，使我們只需處理新增資料而不用再度掃描原有資料庫即可重建 all-subset tree 。此外，當有商品類別新增或剔除時，如何維護 all-subset tree 也是一個值得研究的議題。

致謝

本文作者感謝兩位評審的寶貴意見，與教育部卓越計劃「知識經濟與電子商務之整合性研究」補助。

參考文獻

1. 陳彥良、凌俊青、許秉瑜， 2001 『在

包裹式資料庫中挖掘數量關連規則』，資訊管理學報，第七卷・第二期：215間C

2. 沈清正等，2002『資料間隱含關係的挖掘與展望』，資訊管理學報，第九卷・專刊期：75頁。

3. Agrawal, R. and Srikant, R. "Fast Algorithms for Mining Association Rules," Proc. of the 20th Int'l Conference on Very Large Databases, Santiago, Chile, Sep. 1994.

4. Chen, M.S., Han, J. and Yu, P.S. "Data Mining: An Overview from a Database Perspective," IEEE Transactions on Knowledge and Data Engineering (8:6) 1996, pp: 866-883.

5. Han, J. and Kamber, M. Data mining: Concepts and Techniques, Academic Press, 2001.

6. Han, J., Pei, J. and Yin, Y. "Mining Frequent Patterns without Candidate Generation," Proc. 2000 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'00), Dallas, TX, May 2000, pp: 1-12.

7. Li, S., Shen, H. and Cheng, L. "New Algorithms For Efficient Mining Of Association Rules," Information Sciences (118:1-4) 1999, pp:251-268.

8. Park, J-S., Chen, M-S. and Yu, P. S. "Using a Hash-Based Method with Transaction Trimming for Mining Association Rules," IEEE Trans. on Knowledge and Data Engineering (19:5) 1997, pp:813-825.

9. Pasquier, N., Bastide, Y., Taouil, R. and Lakhal, L. "Efficient Mining Of Association Rules Using Closed Itemset Lattices," Information Systems (24:1) 1999, pp:25-46.

10. Savasere, A., Omiecinski, E. and Navathe, S. "An Efficient Algorithm for Mining Association Rules in Large Databases," Proc. Int'l Conf. Very Large Data Bases, Zurich, Switzerland, Sep. 1995, pp:432-444.

11. Toivonen, H. "Sampling Large Databases For Association Rules," The 22th International Conference on Very Large Databases (VLDB'96), Mumbay, India, Sep. 1996, pp:134-145.