高效率探勘關聯規則之演算法－EFI

黃仁鵬
南台科技大學資訊管理所

藍國誠
南台科技大學資訊管理所

摘要

資料探勘的技術變得日益重要，也廣泛的應用在商業上的預測以及決策的支援。關聯法則在資料探勘的領域中也扮演相當重要的地位，許多關聯法則演算法不斷被提出、改進，以增進效能或節省記憶體空間；本研究也朝著這個目標，試著改進關聯規則演算法為主要方向。

本研究主要是針對探勘關聯規則 QDI 演算法的特性及缺點來加以改進，雖然 QDI 演算法已是很有效率的演算法之一，不過，它還是有兩個最主要的問題。第一，QDI 演算法無法探勘交易長度太長的交易資料庫。第二，QDI 演算法對記憶體使用率不佳；因此，QDI 演算法的實用性大打折扣。

基本於上述理由，本研究提出一個改良 QDI 演算法產生項目集的核心概念新演算法 EFI (An Efficient Approach for Filtering Infrequent Itemsets)。EFI 演算法的特色就是二階段過濾的方式，因為該過濾方式可大量減少非高頻項目集的數量，將更能適用於探勘交易長度較長的資料庫，僅需掃描資料庫四次且不需要產生任何候選項目集，即可快速找出關聯規則。另外，EFI 演算法也改進 ICI-like 演算法因儲存大量項目集須耗用龐大記憶體空間的缺點，每筆交易經過二階段過濾機制的處理後，僅會產生最有可能成為高頻的項目集，因此，EFI 能大量降低項目表須耗用的記憶體空間，以提升記憶體的使用率。在現實生活中的資料庫容量通常都是大於記憶體容量，為了解決此問題，EFI 演算法將選擇採用資料庫分割方式繼續執行探勘任務，每個子資料庫僅需四次 I/O 動作，不隨著高頻項目集的長度增長而增加 I/O 次數，以避免耗費過多的 I/O 時間，也可有效提高執行效率與實用性。

關鍵字：資料探勘，關聯法則，二階段過濾

An Efficient Algorithm for Mining Association Rules–EFI

Jen-Peng Huang
Department of Information Management Southern Taiwan University of Technology

Guo-Cheng Lan
Department of Information Management Southern Taiwan University of Technology

Abstract

The technology of data mining is more important in recent years, and it is generally applied to commercial forecast and decision supports. Association rules mining algorithms in the field of data mining play the important role. Many of association rules mining algorithms were proposed to improve the efficiency of data mining or save the utility rate of memory. So, our major study tries to improve the efficiency of association rules mining algorithms.

In this paper, our major study is to improve the defects of the QDI algorithm. Although QDI algorithm was one of the most efficient algorithms, but it still has two serious problems; in the first place, QDI algorithm can't mine the transactions of databases whose record length is very long; in the second place, QDI algorithm isn't very efficient at utility rate of the memory. Therefore, the QDI algorithm is not very practical.

Based on above reasons we propose a new algorithm – EFI (An Efficient Approach for Filtering Infrequent Itemsets) that is improved from QDI algorithm. The one of the characters of EFI algorithm is the two phrase filtrations which can reduce lots of non-frequent itemsets and is very suitable to mine the transactions of databases whose record length is very long. To find association rules quickly the EFI algorithm only scans database four times and doesn't generate any candidate itemset in mining process. Besides, the EFI algorithm also improves the defect of the ICI-like algorithms that need lots of memory spaces to store lots of sub-itemsets which are decomposed from the transaction records of database. However, the EFI algorithm uses the two phrase filtrations to filter out lots of non-frequent itemsets; it only generates the itemsets which are the most possible to be the frequent itemsets. So, the EFI algorithm can decrease a large number of non-frequent itemsets and increase the utility rate of memory. The size of the databases in the real world is always greater than the size of the memory. In order to solve this problem, the EFI algorithm divides a large database into many sub-databases and mines association rules from those sub-databases. The EFI algorithm only scans database four times and will not be affect by the length of frequent itemsets. The EFI algorithm avoids wasting a lot of I/O time and increases the efficiency and the practicability in application.

Keywords: data mining, association rules, two phase filtrations

壹、緒論

關聯規則在資料探勘領域裡是非常重要的一部份，其中以 1994 年 Agrawal 所提出的 Apriori 演算法(Agrawal & Srikan, 1994)最具代表性。Apriori 演算法不斷利用前一階段的高頻項目集來產生下一階段的候選項目集，再一一去計算支持度，然後依最小支持度與最小信賴度作為門檻值，過濾非高頻項目集進而推導出關聯法則。這種方式雖然在邏輯上很簡單，但要多次掃描資料庫，因而導致效能不佳，因此，許多學者依其缺陷提出了許多改良方法，主要目標是在減少資料庫的掃描次數與增加執行的效能，這些方法皆有相同共通點，也就是避免產生大量的候選項目集(Candidate Itemsets)，再進行評估候選項目集是否通過最小支持度(Minimum support)與最小信賴度(Minimum confidence)等門檻值，以找出符合使用者所需之關聯規則(Association Rules)；像是 DHP 演算法(Park & Che, 1995)可以減少候選項目集，DIC 演算法(Brin, Motwani, Ullman, & Tsur, 1997)可同時掃描多個階段，降低掃描資料庫的次數，以及 FP-growth 演算法(Han, Pei, & Yin, 2000)，其利用 FP-tree 資料結構來存放交易項目，即將交易資料壓縮在樹狀結構中，且只需要掃描資料庫兩次，在探勘過程中也不需要產生任何候選項目集，可大幅加快資料探勘的速度，但缺點是資料結構較複雜，實作不易。爾後陸續又有許多演算法針對降低資料庫掃描次數、減少產生候選項目集的數量、或是完全不用產生候選項目集的方式提昇整個執行效能，像是 FPL 演算法(Tseng & Hsu, 2001)、ICI 演算法(黃仁鵬、錢依佩、吳聲弘，民 92)、QSD 演算法(黃仁鵬、陳秀如，民 93)、QDI 演算法(黃仁鵬、黃南傑，民 93)、IDA 演算法(黃仁鵬、熊浩志、郭煌政，民 93)、GDA 演算法(黃仁鵬、熊浩志，民 94)…等。本研究所提出之新演算法 EFI(An Efficient Approach for Filtering Infrequent Itemsets)，在執行過程中，不需要產生任何候選項目集，只需要掃描資料庫四次，第一次將得到高頻 1-項目集，第二次將刪除資料料庫中非高頻項目，第三次找出高頻 2-項目集，最後，第四次將直接進行探勘處理，在整個執行程序中，不需要複雜的資料結構，即可完成整個關聯規則探勘。

本研究主要是提出一個新的關聯規則演算法 EFI，總共分為 5 章節。在本研究第 2 章節中，介紹關聯規則的演算法相關文獻。在第 3 章節中，提出本研究所發展的新演算法─EFI 演算法相關內容，並在第 4 章節以一實例，說明 EFI 的整個運作過程，另外，在第 5 章將比較 EFI 與其它各演算法的效能評估。最後，在第 6 章節對本研究做結論。

貳、文獻探討

一、Apriori 演算法

在現今研究關聯法則裡，最具代表性的方法就是由 Agrawal 等人於 1994 提出的 Apriori 演算法(Agrawalet al., 1994)，其執行過程主要分為兩步驟：(1)逐層擴展找出所有高頻項目集。(2)透過分析高頻項目集以建立合適之關聯規則。雖然 Apriori 演算法的運作架構簡單，但仍有兩個重要的瓶頸，其內容如下：

(1) 產生大量的候選項目集(Itemset)

在產在 2-候選項目時是由 1-高頻項目集兩兩合併產生，若 1-高頻項目集中有 k 個項目，共會產生(k-1)+(k-2)+…+1 個 2-候選項目，即 k*(k-1)/2 個。假設 1-高頻項目集中有 1000 個項目，則產生 45 萬個 2-候選項目。

(2) 需要多次掃描資料庫

由(1)結果可知，因為有大量的候選項目，而且每一個項目都必須掃描整個資料庫求取其支持度(support)，造成整體執行效率不佳。所以，本研究的目的在於改善，產生高頻項目集時所需花費的時間。

二、FP-growth 演算法

FP-growth 演算(Han et al., 2000)是由 Han 等人所發表的新理念，也就是「不必產生候選項目集方式」亦可完成高頻項目集的探勘任務；它不再利用 Apriori 以分階段產生候選項目集的方法，因為此類的作法將必須不斷反覆地掃描資料庫且產生過多的候選項目集，所以，使整個執行過程相當耗損時間，尤其是磁碟 I/O 的時間，影響甚巨。而 FP-growth 演算法主要分為二階段，僅需掃描資料庫兩次。其中，第一次先將交易資料庫掃描過一次後，將每個支持度大於或等於最小支持度的項目找出，並依據支持度值的大小和在資料庫出現的先後次序作排序，並產生一個 Header table；而第二次掃描時，將過濾交易中不足最小支持度的項目，並依據 Header table 的次序得到每筆交易的高頻資料項目型樣，之後再依據高頻資料項目型樣建構 FP-tree 結構，再利用 Header Table 來找出 conditional pattern FP-tree，並以遞迴方式找出所有的高頻項目集；而 FP-growth 演算法的優點為，不必產生任何候選項目集，可將資料庫壓縮在 FP-Tree 結構中，也改進了多次掃描資料庫的次數，但其缺點為，所運用到的資料結構較為複雜，較不易實作。

三、ICI 演算法

ICI (Incremental Combination Itemsets)演算法(黃仁鵬等，民 92)是由錢依佩等人於

2002 年所提出的，ICI 主要是改進傳統演算法產生候選項目集的方式。它利用交易記錄直接對映 MAP 模組的方式來產生所有的項目集組合。首先將資料庫中的交易資料與 MAP 模組讀取出來，並一一與 MAP 模組對映，以產生該筆交易的所有子項目集，然後再將所得到的項目集都存入項目表中。所以，ICI 演算法在掃描資料庫一次後，就得到所有的項目集組合。最後，再由使用者輸入最小門檻值與最小信賴度，即可產生所需的高頻項目集與關聯規則。

雖然 ICI 演算法可改進傳統產生候選項目集的演算法，但本身最大的缺點其實就是模組的部份，因為模組產生需要耗費相當多的時間，且模組也須事前產生，否則在探勘過程中，才進行產生新模組，將會影響到整個探勘的效率。在漸增式探勘方面，ICI 演算法因完全不事先將非高頻項目集刪除，將導致 ICI 演算法耗費非常大量的記憶體，以及在探勘交易長度較長的資料庫時，都很容易發生記憶體不足的情況。

四、QSD 演算法

QSD 演算法(黃仁鵬等，民 93)是由陳秀如等人於 2004 年所提出的，它其主要是針對 ICI 演算法(黃仁鵬等，民 92)的缺點加以改進。其中，QSD 演算法最主要的改進就是去除了 ICI 演算法中的模組觀念，直接以商品項目本身來作拆解，減少了交易資料與模組間的對映過程。

由於 QSD 演算法是針對 ICI 演算法中使用模組對映的方式加以改進，並以「新增」和「取代」的方式建立二元樹且同時產生所有子項目集的方式來取代 ICI 的模組，所以，QSD 在效能表現上比起 ICI 演算法又更加有效率。另外，雖然 QSD 也具備可做漸進式探勘、變動支持度後不需重新掃描等特性。但也因為 QSD 未刪除任何的非高頻項目集，所以，QSD 將與 ICI 演算法同樣會耗費大量記憶體空間，且無法探勘交易長度較長的資料庫。在實作方面，由於 QSD 演算法必須依賴程式語言中，所提供效能還不錯的 replaceAll 函數，所以，若要以其它的程式語言實作時，將可能會發生因為找不到相同功能的函式，導致 QSD 演算法無法順利實作出來。

五、QDI 演算法

QDI 演算法(黃仁鵬、黃南傑，民 93)是 ICI 演算法(黃仁鵬等，民 92)之改進，主要是改良當交易長度越長時，ICI 演算法的模組對映取代將花費更多的時間，主要原因為當交易記錄長度越長時，模組所存放的項目相對就越多，因此，在整個過程中，所要對映取代的次數也就大幅的增加，當然在執行效率方面，也就大幅的降低。

由於 QDI 演算法是以字串取代的方式來產生所有的子項目集，雖然改善了 ICI 原本使用模組對映的方式，且在效能方面，比 QSD 稍微要好一些；然而，QDI 演算法同樣也承襲了 ICI 與 QSD 演算法相同的缺點，在此就不贅述。

六、IDA 演算法

IDA 演算法(黃仁鵬、熊浩志、郭煌政，民 93)是由熊浩志等人於 2004 年所提出，其主要是針對 ICI、QSD 以及 QDI 演算法的特性及缺點來加以改進，其中，IDA 的直覺拆解方式主要是用來取代原本在 ICI 演算法中的模組映對的拆解方式，而直覺拆解方式就是利用簡單的二進位觀念來達到所要的結果，也就是當讀取到一筆長度為 n 之交易時，只要去產生由 1~2n-1 的二進位數字與原始交易項目做遮罩，即可產生所有的項目集。

雖然 IDA 已經大幅改良之前 ICI-like 演算法的效能，但仍然有兩個較大的缺點，首先，IDA 將會受限於 int 型態(32-1bits)所能表達的範圍，也就是當交易長度大於(32-1bits)時，將無法以 int 型態來進行探勘；另一方面，在記憶體使用方面，雖然 IDA 已先去除掉交易資料中的非高頻的項目，但仍必須產生出全部的項目集，所以，仍會耗用相當多的記憶體，即使記憶體空間放得下，對於記憶體的使用率仍然太差；因此，當 IDA 演算法遇到記憶體不足的情形時，效能也將會大打折扣，或是無法完成探勘動作。

七、GDA 演算法

GDA 演算法(黃仁鵬、熊浩志，民 94)是由熊浩志等人於 2004 年所提出，其主要是針對 IDA 演算法的兩大缺點來加以改進。其中，在 GDA 演算法中，最重要的步驟就是階段拆解的部份，即從可能的範圍內去做階段拆解的動作，例如目前處理到長度 10 時，只要將大於長度 10 的交易做階段拆解的動作即可；另外，在拆解的過程，採用與 IDA 不同的遮罩模式來產生子項目集，且當該階段已無高頻項目集時，即可直接停止後續的探勘動作。

由於 GDA 演算法是採用階段性拆解方式進行運作，所以，GDA 的優點為，可在每一階段適時的釋放記憶體空間，以提升記憶體的使用率；然而，儘管 GDA 已先將交易依長度分別存放，但仍然必須進行多次掃描資料庫的動作，將會耗用大量時間在進行 I/O 部份，使得整個執行效能不佳。

參、EFI 演算法

EFI 演算法提出的目的，就是欲改善 ICI-like 等演算法的缺點，雖然上述相關演算法的效能已經不錯了，但仍有些缺點須待改進；首先，ICI-like 演算法在拆解交易長度較長的交易時，會產生大量的項目集數量，將導致整個執行效能下降或浪費記憶體空間等情況發生；另外，因為 ICI-like 演算法皆無過濾項目集的機制，因此，無法避免大量非高頻項目集的產生，在記憶體利用率上是相當低的，所以，在整個執行效能與記憶體利用率上都須加以改進。

有鑑於 ICI-like 演算法的缺點部份，所以，本研究所提出了一新的演算法為 EFI 演算法，EFI 演算法仍會保留 ICI-like 演算法的快速產生項目集核心概念，把效能不好的地

方加以改進。其中，ICI-like 演算法所提出之項目表的概念，本研究將會繼續採用且加以改良其儲存的效能，以提升整個存取的速度；另外，在產生項目集的部份，本研究僅將產生項目集的過程修改成更簡單的直接配對產生方式來達成。至於漸增式探勘（Incremental Mining）的部份，由於 ICI-like 等演算法都是將所有的交易記錄進行完整拆解，再儲存於項目表內，因此，只要探勘完成後，將項目表中的資訊加以儲存，爾後，若資料庫內容有更動時（增加或減少），只要針對變動的部份來做拆解，並將拆解完的結果存入（或刪除）至項目表即可完成漸增式探勘。然而，諸如此類的作法，將無法儲存資料庫中長度較長的交易內容，因為光是一筆長度為 20 的交易記錄，就會拆解出 2^{20}-1=1,048,575 個子項目集，將會耗費大量的記憶體空間；因此，在探勘過程中，EFI 演算法會先進行第一階段的過濾，也就是先把整個資料庫裡的交易記錄，將低於最小支持度的項目予以刪除，使每筆交易長度縮短，並將相同記錄合併累加次數，可避免不斷重複拆解相同交易記錄，接著，找出高頻 2-項目集且保留在一暫存項目表中，再利用高頻 2-項目集作為後續產生項目集過程時，所有項目間的頻繁關係核對，此作法將可有效避免產生大量的非高頻項目集；另外，在進行產生新項目集之前，將會先進行判斷是否有高頻 2-項目集的存在，若有高頻 2-項目集存在時，則繼續進行執行後續的探勘動作；若沒有高頻 2-項目集存在時，則直接結束後續的探勘動作，因為，在資料庫中已無法再找出大於高頻 2-項目集長度以上的高頻項目集，如此將可省下非必要的探勘時間。

EFI 演算法的流程圖，如圖 1 所示；EFI 演算法的整個運作過程，將分別在第三章中的各小節會詳細說明；首先，在第三章第一小節中說明資料壓縮，而第二小節將說明如何縮短交易長度及合併處理，接著將在第三小節中介紹減少非高頻項目集的程序，並於第四小節與第五小節中，將針對產生項目集與減少非高頻項目集產生的程序進行說明，最後，將簡單地說明項目表於第三章第六小節。

一、資料壓縮

EFI 演算法為了能減免記憶體的負擔與配合項目表(Itemset Table)存取的型態 char[]，所以，在整個資料庫讀入到記憶體時，將會對讀入的資料庫進行資料替代，藉以對資料進行壓縮，以減免記憶體的負擔；換言之，也就是利用 char 的型態取代每一個商品項目，另外，如果項目數大於 char 的上限時，也可考慮以其它型態來替代，如 int 基本型態等，由於資料壓縮部份是彈性的，並無固定非得以 char 模式進行資料壓縮，而在整個演算法架構上，可隨著項目數的大小而定；此作法可使記憶體的使用更有效率，在執行效能方面也有所助益，因有更多的記憶體空間可使用，可有效提升整個演算法的執行效能。

二、縮短交易長度及合併處理

EFI 演算法為了能夠適用於交易長度較長的資料庫探勘，將加入縮減交易長度的方式。首先，依序將資料庫讀入到記憶體裡，同時也將計算各個交易項目的出現次數，因此，當讀取完資料庫後，即可得到所有項目及各自的支持度，緊接著篩選未通過最小支

持度之交易項目，將資料庫中的非高頻項目予以扣除，此方式可將非必要的項目事先刪除，以減少交易的長度，對後續的執行處理與儲存空間皆有很大的幫助。這就是本研究中的第一階段有效縮減交易長度的部份，此方式與 FP-growth 演算法類似，也就是事先刪除資料中未通過最小支持度的項目，但不同之處在於 FP-growth 演算法會將每筆交易資料重新依照支持度排序，在 EFI 演算法刪除過程中並無排序動作。

當交易經過縮短交易長度後，將會產生多筆相同的交易記錄，因此，本研究為了避免重複拆解相同交易記錄及耗費不必要的時間成本，將每筆交易經過縮短交易長度後，進行儲存及合併相同交易記錄及筆數，也就是當有多筆相同的交易記錄時，僅記錄交易內容及筆數。本研究為了有效降低合併處理所需耗費的時間成本，將參考於 java API 的 HashMap 加以修改而成的項目表，主要原因為加以改良後的項目表，其儲存結構也可適用於儲存交易記錄與次數，而非侷限於儲存項目集與支持度，因此，本研究將相同的儲存結構運用在縮短交易長度及合併處理階段，將可有效提升該階段的儲存效能，可避免因執行合併處理而浪費過多非必要的時間成本。

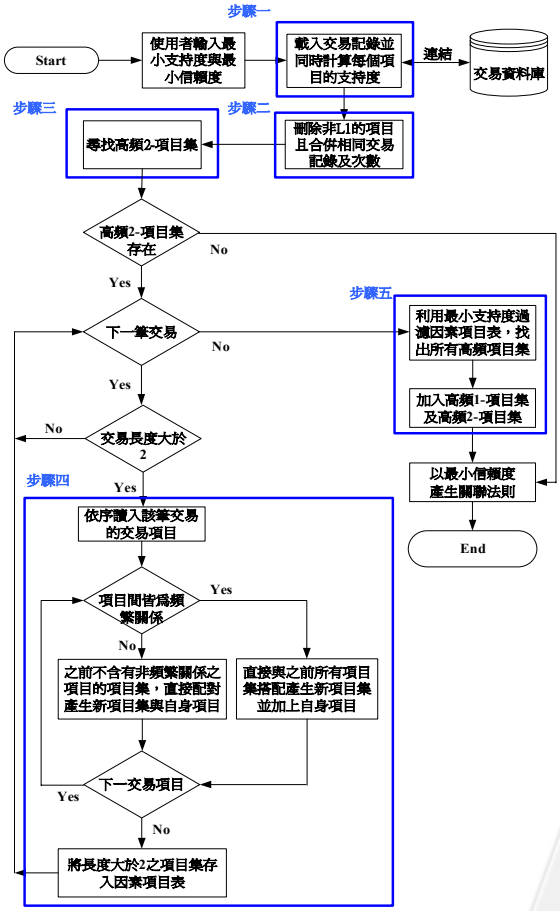

圖 1：EFI 演算法流程圖

三、減少非高頻項目集的程序

減少非高頻項目集產生方式主要是用來取代原本在 QDI 演算法中的拆解置換方式，以提高產生新項目集的效能。在 QDI 演算法的置換功能，其就是將某個長度為 n 項目集的所進行 n-1 次置換動作，即可得到所有的子項目集。本研究發現其實要產生 n-項目集的所有子項目集，並不須要利用模組對映或置換等方式，只要簡單的直接配對概念即可達到一樣的結果，此方式也較適合運用在本研究之減少非高頻項目集產生之部份。所謂簡單的直接配對觀念就是當讀取到一筆長度為 n 之交易時，只要將目前該項目與之前所產生之項目組合，直接進行搭配將可產生新項目集，再加上自身項目，即可產生所有該交易項目之項目組合，將可省去使用模組對待或置換等方式所需多花費之時間成本；在直接配對的過程中，也將判斷該項目與之前每一項目之間是否具有非高頻關係，若成立，將不與之前所有項目集裡含該項目配對產生新項目集，因為兩項目所產生之所有項目集必定為非高頻項目集；反之，將直接進行產生該項目與之前所有項目集配對產生新項目集並加上自身項目。由於使用這種減少非高頻項目集方式後，也使得整個產生項目組之流程更加快速，在效能方面也大幅提昇。

四、產生項目集的程序說明

(1) 假設目前交易長度為 1 時，則會產生出 $1\sim2^1-1=1$ 個項目集，因此，所有的子項目只有自己本身而已，如圖 2 所示：

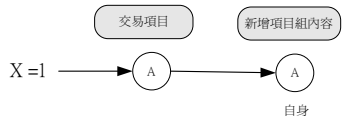

圖 2：交易長度為 1 之項目集產生過程圖

(2) 假設目前交易長度為 2 時，則會產生出 $1\sim2^2-1=3$ 個項目集，因此，所有的子項目集與產生過程，如圖 3 所示：

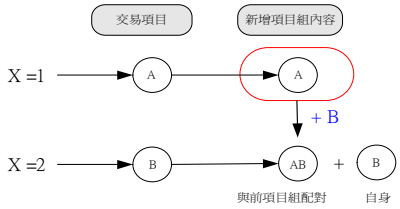

圖 5：交易長度為 2 之項目集產生過程圖

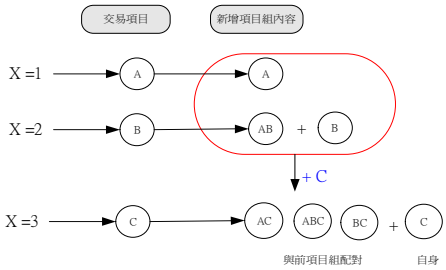

圖 4：交易長度為 3 之項目集產生過程圖

(3) 假設目前交易長度為 3 時，則會產生出 $1\sim2^3-1=7$ 個項目集，因此，所有的子項目集與產生過程，如圖 4 所示。

(4) 依此類推長度為 n 時，只要將交易依上面直接搭配，即可產生出長度為 n 的所有子項目集。

以上為 EFI 演算法新增項目集方式之說明，只要根據以上的方式以此類推，即可求出所有長度 n 的子項目集。

五、減少非高頻項目集的程序說明

假設某筆交易長度為 4 且內容為{ABCE}，若高頻 2-項目集為{AB}與{AE}，減少非高頻項目集的過程如圖 5 所示。首先，由於第一項目 A 之前並沒有任何項目，故直接產生項目集 A；接著，第二項目 B 與 A 具有頻繁關係，所以，B 與之前所有含有項目 A 的項目集配對產生新項目集，故新項目集 AB 與項目 B 本身；接下來，第三項目 C 與前兩項目 A 與 B 皆無頻繁關係，故只加入自身項目 C；最後，第四項目 E 與 A 因具有頻繁關係且與 B、C 具有非頻繁關係，所以，項目 E 只與之前所有項目集含有 A 項目且不含 B 與 C 的項目集配對產生新項目集，因此，產生新項目集 AE 及自身 E，最後，將得到該筆交易之所有可能高頻項目集 A、B、AB、C、AE 與 E 等六個項目集，此過濾方式不必產生所有的全部項目集，將可以避免產生大量具有非頻繁關係的項目集。

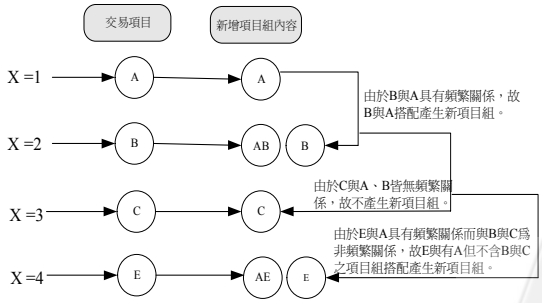

圖 5：減少非高頻項目集產生過程

```
class Element {
      final char[] key;        //元素的 key 值(用來存放項目集)
      int value;               //元素的 value 值(用來累計其支持度)
      int hash;                //元素的 hash 值
      Element next;            //相同 hash 值的下一個元素

      //Some of Methods
}
```

圖 6：項目表的元素資料結構

六、項目表

項目表(Itemset Table)是用來存放所有項目集的組合，最主要目的是能快速篩選出通過最小支持度的項目集，並將通過最小支持度的高頻項目集輸出。在實作項目表時，本研究則是參考 java API 中的 Hash Map 所修改而來的；會採用 Hash Map 的主要原因在於 Hash Map 的主要概念是利用 key 來找尋其對應的 value，符合本研究的項目表所需要儲存的「項目集」與「支持度」的資訊，另外，本研究也在儲存的格式上也稍作改變，將原先 Hash Map 的 Object 型態修改成 key(char[])對應到 value(int)，除了可節省並更有效利用記憶體空間外，也避免儲存「支持度(相對 Hash Map 的 value)」所須進行轉型的時間，所以，直接改採用 int 的基本型態，將可節省大量的轉型時間，這些因素都是有效能上的考量。因此參考 Hash Map 的架構來實作項目表，命名為 Itemset Table。以下簡單說明項目表的架構。

如圖 6 內容所示，Itemset Table 大致上與 Hash Map 差不多，只是將其中每個元素的資料結構以更符合項目表的需求來設計，並額外提供重要的 add(itemset, count)功能用來快速累加項目的支持度。

肆、完整實例說明

本章範例將依據 EFI 演算法的流程步驟，說明整個 EFI 演算法的執行程序。在步驟一中，將先載入資料庫中的交易記錄，並進行資料空間的壓縮處理，也就是將交易記錄項目轉為 char[]型態，而資料壓縮方式如同第三章第一小節內容所述，可節省記憶體空間的使用，執行過程如圖 7 所示。接著，將進行步驟 2 的流程，而縮短交易長度及合併處理如同第三章第二小節內容所述，執行過程如圖 8 所示。資料庫經過步驟一及二之程序後，再以新資料庫取得高頻 2-項目集，執行過程如圖 9 所示。因為已取得高頻 2-項目集的過濾資訊，所以，開始依序產生每筆交易可能為高頻的項目集，並存入項目表內，而減少非高頻項目集程序如同第三章第三小節及第五小節內容所述，執行過程如圖 10 至 15 所示。最後，再以最小支持度從項目表得到高頻項目集，並存入高頻項目表，同時，也將高頻 1-項目集及 2-項目集一起存入高頻項目表，將可得到該資料庫的所有高頻項目集，執行過程如圖 16 所示。

Step1：資料壓縮

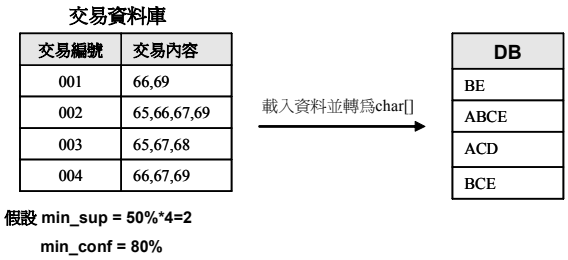

圖 7：步驟一之資料壓縮過程圖

Step2：縮短交易長度及合併處理

圖 8：步驟二之縮短交易長度及合併處理過程圖

Step3：尋找高頻 2-項目集資訊

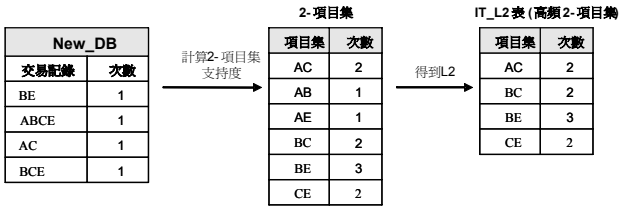

圖 9：尋找高頻 2-項目集過程圖

Step4-1：產生項目集且存入因素項目表內(讀入第一筆交易記錄 {BE})

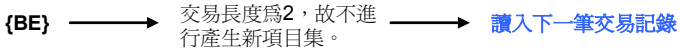

圖 10：第一筆交易記錄處理過程

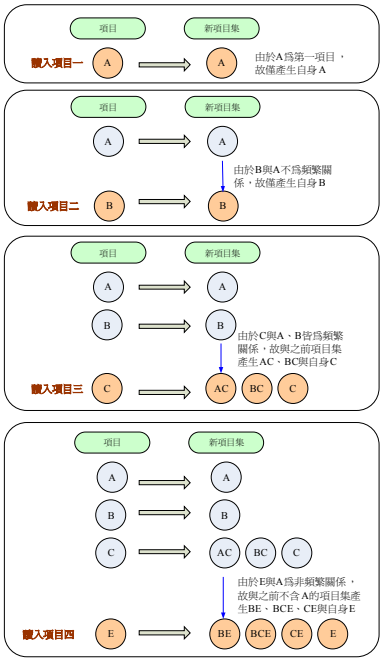

圖 11：第二筆交易記錄產生新項目集過程

Step4-2：產生項目集且存入因素項目表內(讀入第二筆交易記錄 {ABCE})

第二筆交易

新增項目集			
項目集	次數	項目集	次數
A	1	BE	1
B	1	BC	1
C	1	CE	1
E	1	BCE	1
AC	1		

將長度大於2的項目集存入項目表

IT 項目表

項目集	次數
BCE	1

圖 12：將長度大於 2 之項目集存入 IT 項目表

Step4-3：產生項目集且存入項目表內(讀入第三筆交易記錄 {AC})

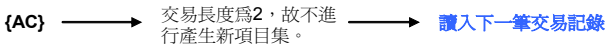

圖 13：第三筆交易記錄處理過程

Step4-4：產生項目集且存入項目表內(讀入第四筆交易記錄 {BCE})

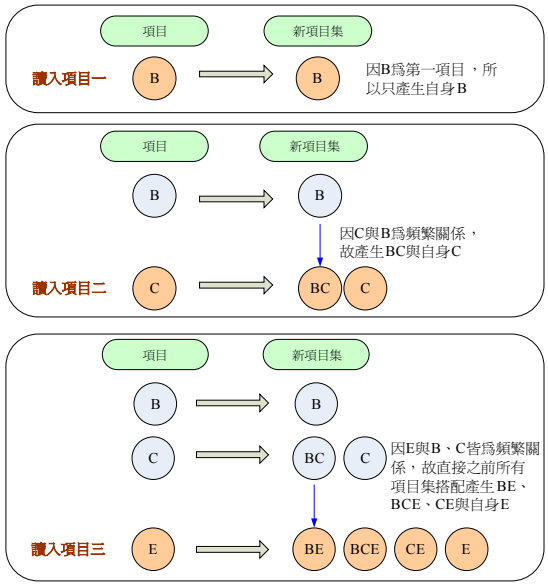

圖 14：第四筆交易記錄產生新項目集過程

第四筆交易

新增項目集	
項目集	次數
B	1
C	1
E	1
BE	1
BC	1
CE	1
BCE	1

將長度大於2的項目集存入IT

IT 項目表

項目集	次數
BCE	2

圖 15：將長度大於 2 之項目集存入 IT 項目表

Step5：得到所有的高頻項目集

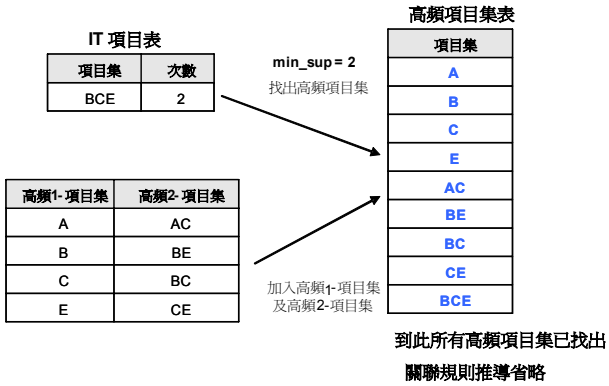

圖 16：得到所有高頻項目集之過程

伍、效能評估

一、實驗環境

CPU： Intel Pentium IV 2.8GHz
RAM： 512MB DDR RAM
OS： Windows 2000 Server
資料庫來源： 由 IBM Data Generator 所產生的資料
開發工具： J2SDK 1.4.2

二、資料庫參數說明

本研究實驗為求公正，實驗資料庫是由 IBM Data Generator 所產生；另外，本實驗所使用到的相關設定參數，如下說明：

T：每筆交易的平均長度。 (對應到的參數 -tlen)
I：潛在高頻項目集的平均長度。 (對應到的參數 -patlen)
N：資料庫中商品項目的種類。 (對應到的參數 -nitems)
D：資料庫的資料筆數。 (對應到的參數 -ntrans)

而其它未加以說明的參數均使用預設值。另外，由於真實資料庫來源不易取得，因此，本研究採用微軟公司的 Microsoft SQL Server 2000 所附之 FoodMart(微軟公司，2000)跨國連鎖零售商店之範例資料庫來模擬真實交易資料庫的環境，且經過彙整後，再分別

隨機抽取 5,000、10,000、15,000、20,000 與 25,000 筆交易記錄進行實驗模擬，以測得各演算法在模擬真實資料庫中的執行效能。

三、測試的演算法

本研究提出一個從 ICI-like 演算法產生項目集的核心概念加以改良而來的新演算法 EFI，所以，將具有相同核心概念的 QSD、QDI 與 IDA 等演算法，一起納入本研究比較之演算法裡。另一方面，也將 QDI 與 QSD 等演算法加入與 IDA 演算法相同之過濾機制，即在執行過程中，將資料庫每筆交易中的非高頻項目予以刪除，此方式可有效縮短交易長度，至於 QDI 及 QSD 演算法其它部份則是完全按照原先的設計架構，此修改的主要原因在於 EFI 演算法本身不具有漸增式探勘的功能，所以，此作法將更能符合與 EFI 演算法進行比較的公平性；除此之外，本研究也將傳統有名的 Apriori 演算法、FP-growth 演算法及針對 IDA 演算法加以改良的 GDA 演算法，皆一同納入本研究實驗模擬的比較演算法裡。所有演算法皆使用與 EFI 演算法相同的項目表，將可更公平測出各演算法的核心的效能。在其它方面，每個演算法皆是以 Java 語言進行開發，且在同一台機器上進行效能比較，藉以排除其它因素的不同而導致執行效速不一樣。

接著將針對所實作比較的演算法其公正性作說明，也就是針對非常著名的 Apriori 及 FP-growth 這兩個演算法來做驗證。驗證的方式則是以 Han 等所發表的文獻(Han et al., 2000)中提到的數據來做比例的推算，圖 17 即為其中出現的數據圖。圖 17 是在資料庫為 T25I20N10K 且 min_sup 為 1.5%的情況下進行測試，測試環境為 Pentium 450MHz CPU、128MB RAM、在 Windows NT 平台以 Visual C++ 6.0 撰寫。而本研究則是以 J2SDK 1.4.2 撰寫這兩個演算法，且在 Pentium IV 2.4GHz、256MB RAM 及 Windows 2000 Server 平台環境下進行測試，也以 IBM data generator 產生一樣參數的模擬資料庫，測試結果如圖 18 所示。我們可將圖 17 及圖 18 的結果作比較，由於使用的語言及測試環境都不一樣，因本研究採用 2.4GHz 的測試環境而 Han 等則是 450MHz 的測試環境，大約差了 5.33 倍。而本研究中實作之 Apriori 及 FP-growth 均有達到此倍數該有的執行水準，甚至更好。雖然這種比較方式可能不夠客觀，但是至少可看出本研究在實作上，確實已儘可能以最佳方式來實作 Apriori 及 FP-growth 演算法，並無故意以較差方式實作他人演算法的顧慮。因此，在後續實驗中，這兩個演算法所測出的數據仍有一定的公正性及可信度。

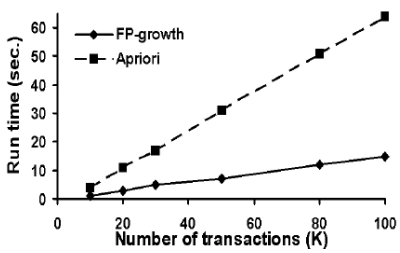

圖 17：FP-growth 與 Apriori 比較圖(來源 Han et al., 2000)

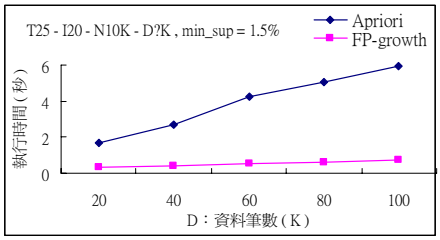

圖 18：本研究實作之 FP-growth 與 Apriori 比較圖

四、實驗設計

本研究共設計九項實驗，其中有五項實驗分別用來測試五種不同的參數設定下對每個演算法效能的影響。同時，本研究也加入各演算法對模擬真實資料庫 FoodMart 之執行效能影響。另一方面，本研究納入比較 EFI 演算法與過去的 ICI-like 演算法儲存於項目表內的項目集數量差異，也加入比較各演算法的記憶體利用率之差異。最後，本研究也模擬當大型資料庫無法一次完全讀入實際記憶體時，EFI 演算法與 Apriori 演算法在效能上的差異。在每個實驗的測試方法裡，本研究只變動所要測試的參數且將固定其它的參數不變；另外，當演算法的效能差異太大時，可能會分為兩部份來測試，第一階段所有的演算法都會進行比較，第二階段則是進階測試，也就是將第一階段中效能較不好的演算法予以排除。每個實驗中的「執行時間」計算，皆是從資料庫載入完畢後開始，到找出所有的高頻項目集及個別出現的次數為止，也就是只測量各演算法核心所須花費的時間，並去除掉每個演算法載入資料庫的時間(因為載入的時間都一樣)。

五、實驗分析

（一）實驗一：在不同最小支持度門檻值下，對各演算法效能的影響

在實驗一裡，將測試在不同最小支持度門檻值下，對各演算法效能的影響，同樣的，固定其它參數不變(T3-I2-N1K-D200K)，實驗結果如圖 19 所示。由圖 19 可得知每個演算法的執行時間都隨著支持度的上升而下降。其中，Apriori 演算法在支持度較大時，甚至要比 QDI 演算法及 QSD 演算法要來得好，然而，當最小支持度門檻值愈小時，Apriori 演算法需花費的時間成本呈大幅上升，主要原因在於當高頻項目愈多時，須產生的候選項目集也就愈龐大，導致執行效能不佳。所以，在進階實驗部份，將 Apriori 演算法予以排除，僅繼續測試其它演算法。

在進階實驗裡，進一步把最小支持度門檻值調降更低並測試在不同最小支持度門檻值下，對各演算法效能的影響，同樣的，固定其它參數不變(T3-I2-N1K-D200K)，實驗結果如圖 20 所示。由圖 20 可清楚得知當最小支持度愈大時，每個演算法的執行效能也隨之愈佳，其中，本研究所提出的 EFI 演算法，即使在最小支持度門檻值較低的情況下，

仍較其它演算法要來得佳且平穩，主要原因在於 EFI 演算法因本身的過濾機制，可避免產生大量非必要的項目集，所以，EFI 演算法能以穩定且佳的執行效能進行探勘任務。

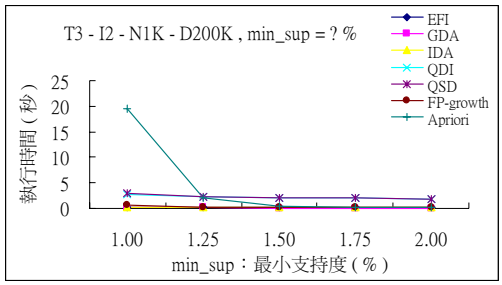

圖 19：在不同最小支持度門檻值下，對各演算法效能的影響圖

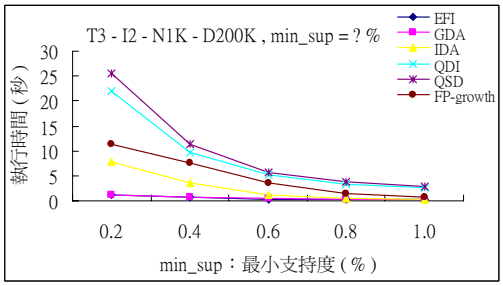

圖 20：在不同最小支持度門檻值下，對各演算法效能的影響圖(進階實驗)

（二）實驗二：在不同參數下，EFI 與 ICI-like 演算法在項目集數量之差異比較

在實驗二部份，將設計三個實驗分別比較 EFI 與 ICI-like 演算法在不同參數下，儲存於項目表內項目集數量與高頻項目集數量之差異。其中，ICI-like 演算法將以執行效能較佳且空間利用率較好的 IDA 演算法作為代表，而 IDA 演算法與過去的 ICI-like 演算法不同之處在於加入先刪除非高頻項目的機制，即先縮短交易長度，再進行交易記錄拆解，最後，再將新項目集儲存於項目表內，所以，本研究計算 ICI-like 演算法儲存於項目表內之項目集數量，將從交易記錄拆解為起始直到整個資料庫拆解完畢後，儲存於項目表內之所有項目集數量皆為計算範圍。EFI 演算法在前三次掃描資料庫時，可得到高頻 1-項目集與高頻 2-項目集等資訊，在第四次則是直接針對每筆交易記錄進行探勘，即僅產生所有可能為高頻的項目集，在計算項目表內項目集數量部份，EFI 演算法將從第四次針對資料庫進行探勘且儲存於項目表內的項目集數量開始算起，其中，也將高頻 1-項目集及高頻 2-項目集納入計算範圍之內，將更能突顯出 EFI 演算法與傳統的 ICI-like 演算法儲存於項目表內的項目集數量之差異。

1. 在不同參數 T 下，EFI 與 ICI-like 演算法在項目集數量之差異比較

在本實驗裡，將比較在不同交易平均長度(參數 T)且最小支持度門檻值為 0.5%下，EFI 演算法與 ICI-like 演算法儲存於項目表內之項目集數量與實際高頻項目集數量的差異，同樣的，固定其它參數不變(I2-N5K-D200K)，實驗結果如圖 21 所示。

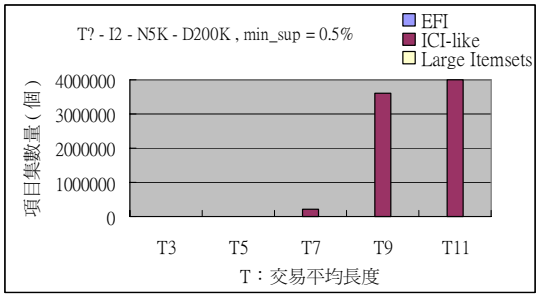

圖 21：在不同參數 T 下，EFI 與 ICI-like 演算法之項目集數量差異圖

由圖 21 可得知當交易平均長度(參數 T)增長時，ICI-like 演算法雖已加入非高頻項目刪除之處理，但因無法有效縮短交易長度，須產生大量的項目集，甚至呈現數量暴增的現象，導致記憶體空間的不足。EFI 演算法因本身的過濾機制，可避免產生大量非必要的項目集，相對的，儲存於項目表內的項目集數量也就相對地接近實際高頻項目集數量，甚至項目表內的項目集皆為高頻項目集時。主要原因在於當最小支持度門檻值非較低時，將可能導致實際高頻項目集的項目僅集中於少數的項目，則 EFI 演算法可利用本身的過濾機制，更精確地得到所有可能為高頻的項目集，因此，儲存於項目表內的所有項目集就可能皆為高頻項目集。

2. 在不同參數 I 下，EFI 與 ICI-like 演算法在項目集數量之差異比較

在本實驗裡，將比較在不同潛在高頻項目集平均長度(參數 I)且最小支持度門檻值為 0.5%下，EFI 演算法與 ICI-like 演算法儲存於項目表內之項目集數量的差異。同樣的，固定其它參數不變(T8-N5K-D200K)，實驗結果如圖 22 所示。

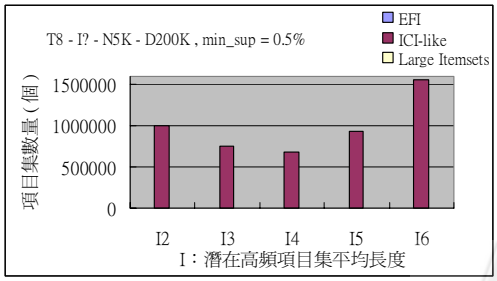

圖 22：在不同參數 I 下，EFI 與 ICI-like 之項目集數量差異圖

由圖 22 可得知當潛在高頻項目集平均長度(參數 I)愈長時，ICI-like 演算法所需產生的項目集數量也就愈多，其主要原因在於當參數 I 愈接近參數 T(交易平均長度)時，代表

該資料庫的緊密性就愈高，不容易在非高頻 1-項目集階段被刪除，導致 ICI-like 演算法無法有效縮短交易平均長度，因此，需產生大量的項目集數量，甚至呈現暴增的情況。EFI 演算法因本身的特殊過濾機制，可在產生新項目的過程中，有效避免產生大量非高頻項目集數量，因此，儲存於項目表內的項目集數量是相當少的，甚至項目表內的項目集皆為高頻項目集。

3. 在不同參數 N 下，EFI 與 ICI-like 演算法在項目集數量之差異比較

在本實驗裡，將進行比較在不同項目種類(參數 N)及最小支持度門檻值為 0.5%下，EFI 演算法與 ICI-like 演算法在項目表內的項目集數量之差異比較；同樣的，固定其它參數不變(T5-I3-D200K)，實驗結果如圖 23 所示。

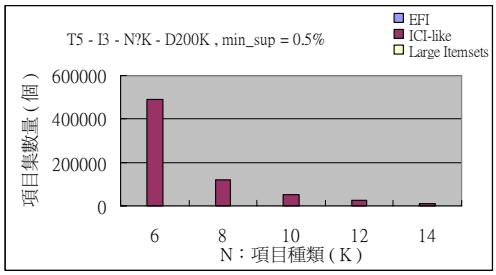

圖 23：在不同參數 N 下，EFI 與 ICI-like 之項目集數量差異圖

由圖 23 可得知當項目種類呈現多樣化時，其被購買的機會也就愈小，因 ICI-like 演算法加入先刪除非高頻項目的處理，所以，當項目種類增多且最小支持度門檻值固定不變時，被刪除的項目將會愈來愈多，ICI-like 演算法將可有效縮短交易的長度，相對的，須儲存於項目表內的項目集數量也就愈少。因項目種類愈多時，EFI 演算法因本身的特殊過濾機制，可更明確地判斷出項目間的頻繁關係，所以，儲存於項目表內的項目集數量將相當接近實際高頻項目集的數量，甚至皆為高頻項目集。

(三)實驗三：在不同資料庫下，各演算法的記憶體使用量差異比較

在實驗三裡，將比較在不同交易平均長度(參數 T)且最小支持度門檻值為 0.5%下，對各演算法所須耗用的記憶體空間差異作比較。同樣的，固定其它參數不變(I2-N10K-D200K)，僅變動參數 T 的設定值，實驗結果如圖 24 所示。

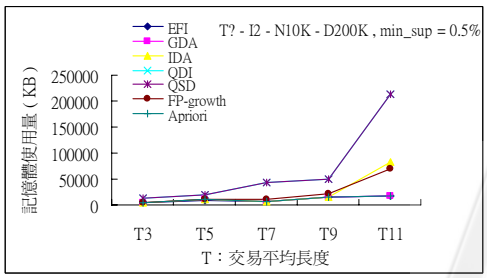

圖 24：在不同交易平均長度下，各演算法的記憶體使用量差異圖

由圖 24 可得知當交易平均長度(T 參數)愈長時，QDI 演算法與 QSD 演算法，因必須拆解出所有的項目集，所以，將導致記憶體使用量暴增，甚至呈現不足的情形；而同樣以 ICI-like 為基礎的 IDA 演算法，因採用字元遮罩產生項目集的模式，所以，在記憶體利用率方面，IDA 演算法較 QDI 演算法及 QSD 演算法以字串為儲存方式為佳。其他演算法則因有適當的過濾機制或本身特有的資料結構，在記憶體使用率上，皆比 ICI-like 演算法要來得穩定且佳。

在進階實驗裡，將繼續比較各演算法在記憶體利用率的差異，因此，將測試在交易平均長度較長且支持度門檻值為 1%下，對各演算法在記憶體利用率上的差異。固定其它參數不變(I2-N10K-200K)，僅變動參數 T，實驗結果如圖 25 所示。由圖 25 可得知當交易平均長度(T 參數)愈長時，各演算法所需的記憶體使用量都呈現增加的情況。其中，ICI-like 相關演算法，因其必須拆解所有的項目集，所以，在記憶體使用量將呈現暴增，甚至發生記憶體空間不足的狀態。FP-growth 演算法因本身的 FP-tree 結構，所以，當交易平均長度小於某一長度下，在記憶體的使用率仍然是不錯的，但當交易平均長度愈長時，其所需的記憶體空間將呈現大幅上揚的狀態，主要原因在於資料可能相對的較為不緊密，造成 FP-tree 需儲存的節點增多，即是其 FP-tree 的分支將呈現廣且深，導致記憶體的需求愈來愈大，甚至呈現暴增現象。Apriori 演算法和 GDA 演算法在執行過程中，皆是採用階段式找出高頻項目集，雖 Apriori 演算法的執行效能較差，但僅保留高頻項目集，所以，在記憶體利用率上是不錯的，而 GDA 演算法雖採用拆解方式，但也是僅保留每一階段的高頻項目集，會適時的釋放空間，因此，在記憶體的利用率上也是不錯的。EFI 演算法因本身的過濾機制，可快速產生所有可能為高頻的項目集，有效避免大量非必要項目集的產生，所以，EFI 演算法在記憶體利用率上，也是有不錯的表現。

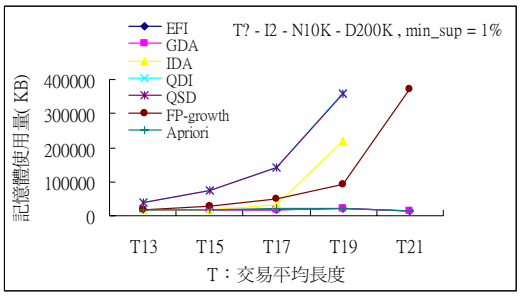

圖 25：在不同交易平均長度下，各演算法的記憶體使用量差異圖(進階實驗)

(四)實驗四：在不同交易平均長度下，對各演算法效能的影響

在實驗四裡，將比較在不同交易平均長度(參數 T)且最小支持度門檻值為 1%下，對不同演算法效能的影響。同樣的，固定其它參數不變(I2-N5K-D200K)，僅變動參數 T 的設定值，實驗結果如圖 26 所示。

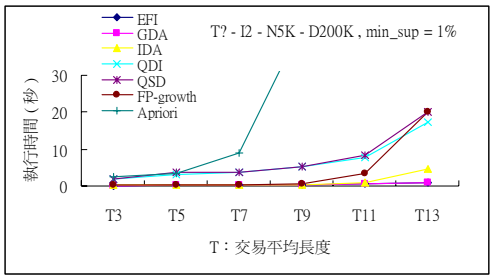

圖 26：在不同交易平均長度下，對各演算法效能的影響圖

由圖 26 可得知當交易平均長度(參數 T)愈長時，各演算法所須花費的時間也隨之增加，主要原因在於當平均交易愈來愈長，所須花費的執行時間與記憶體空間都將同時增加，而導致執行效能較為緩慢且不佳。其中，Apriori 演算法的執行效能較為明顯呈現不佳的狀態，其要原因在於產生候選項目集的過程，因交易平均長度愈長時，其相對必須產生更多的候選項目集數量且掃描資料庫的次數也將增加，所以，其需花費的時間將愈多，甚至呈現暴增的情況。另一方面，ICI-like 演算法將因交易平均長度愈長時，須拆解的項目集也就愈多，所以，其須耗費的時間也將增加，甚至因項目集過於龐大，而造成記憶體不足的情況。當某一交易平均長度及最小支持度門檻值下，FP-growth 演算法因本身的 FP-tree 結構，可有效節省記憶體空間及快速執行探勘任務；然而，當交易平均長度較長且最小支持度門檻值不高時，FP-growth 演算法須花費較多的時間成本，以建立深且廣的 FP-tree 結構，所以，其執行效能將呈現較不佳的狀態。GDA 演算法因使用階段拆解的方式，可快速產生高頻項目集且可在某一階段無高頻項目集時，即停止整個探勘任務，所以，在執行效能方面，GDA 演算法是呈現較佳的狀態。EFI 演算法因本身的過濾機制，可有效避免產生大量非必要的項目集，所以，在執行效能方面，EFI 演算法呈現佳且平穩的狀態。在進階實驗部份，將繼續測試在交易平均長度較長的情況下，對各演算法效能的影響，同樣的，本研究將排除在上實驗內效能不佳的演算法，所以，在進階實驗部份，將僅考慮 EFI、GDA 及 FP-growth 等演算法，納入進階實驗內。

在進階實驗裡，將比較在不同交易平均長度(參數 T)且最小支持度門檻值為 0.25% 下，對 EFI、GDA 及 FP-growth 演算法效能的影響。同樣的，把其它參數固定不變 (I2-N5K-D200K)，只變動參數 T 的設定值，實驗結果如圖 27 所示。由圖 27 可清楚得知當交易平均長度愈長時，每個演算法所需花費的時間也隨之上升，GDA 演算法與 FP-growth 演算法呈現較明顯上揚的狀態，主要原因為當最小支持度門檻值較低且交易平均長度較長時，GDA 演算法將必須產生大量的項目集，而 FP-growth 演算法則因其 FP-tree 結構，須建構深且廣的 FP-tree 結構，而導致須耗費大量的執行效能，所以，GDA 演算法及 FP-growth 演算法在效能方面較不佳。另一方面，EFI 演算法因本身的過濾機制，可避免產生大量非必要的項目集，所以，在執行效能方面，EFI 演算法較其它演算法要來得穩定且佳。

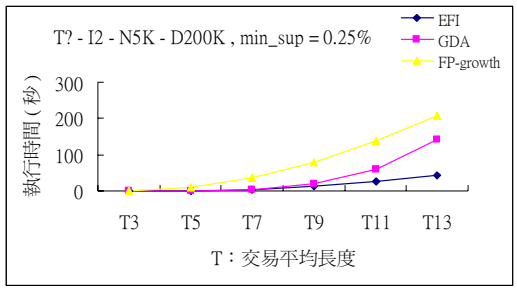

圖 27：在不同交易平均長度下，對各演算法效能的影響圖(進階實驗)

(五)實驗五：在不同潛在高頻項目集平均長度下，對各演算法效能的影響

在實驗五裡，將比較在不同潛在高頻項目集的平均長度(參數 I)且最小支持度門檻值為 1%下，對不同演算法效能的影響。同樣的，固定其它參數不變(T13-N5K-D200K)，僅變動參數 I 的設定值，實驗結果如圖 28 所示。由圖 28 可得知當潛在高頻項目集平均長度(參數為 I)愈長時，在效能方面(特別注意圖的 Y 軸是採用對數刻度)較沒有一定的變化方向；然而，仍可看出 Apriori 演算法明顯比其它演算法需花費更多的執行時間；其它演算法的效能則是呈現不錯的狀態。在進階實驗裡，本研究將調降最小支持度門檻值，因此，QDI、QSD 及 IDA 演算法因無法在較長的交易長度下進行探勘，所以，除了效能較不好的 Apriori 排除，亦將 QDI、QSD 與 IDA 演算法一同予以排除。

在進階實驗部份，將比較在不同參數 I(潛在高頻項目集平均長度)且最小支持度門檻值為 0.25%下，對不同演算法效能的影響。同樣的，固定其它參數不變(T13-N5K-D200K)，僅變動參數 I 的設定值，實驗結果如圖 29 所示。由圖 29 可得知當 min_sup 門檻值較低時，GDA 演算法因須拆解的交易長度也就較長，所以，其所需耗費的時間成本也就增多。當潛在高頻項目集平均長度愈大及最小支持度門檻值較小時，則資料的緊密性將愈高，相同的，也就愈適合 FP-growth 本身的結構，所以，FP-growth 演算法能保有相當不錯的執行效能。另一方面，EFI 演算法因自身的特殊過濾機制，可快速找出所有可能為高頻的項目集，以避免產生過多非必要的項目集，所以，可明顯看出 EFI 演算法較其它演算法要來得佳且穩定。

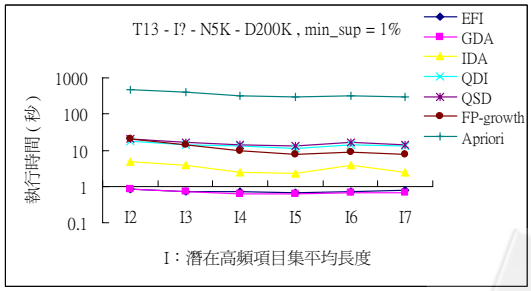

圖 28：在不同潛在高頻項目集平均長度下，對各演算法效能的影響圖

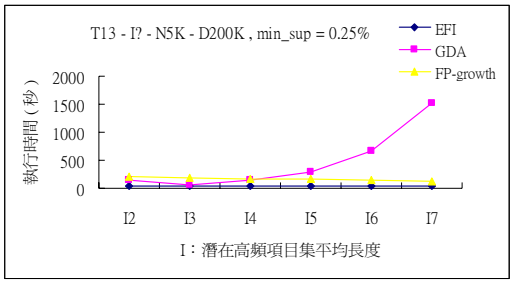

圖 29：在不同潛在高頻項目集平均長度下，對各演算法效能的影響圖(進階實驗)

(六)實驗六：在不同項目種類數量下，對各演算法效能的影響

在實驗六裡，將比較在不同項目種類數量且最小支持度門檻值為 0.5%下，對不同演算法效能的影響。同樣的，固定其它參數不變(T13-I2-D200K)，僅變動參數 N 的設定值，實驗結果如圖 30 所示。

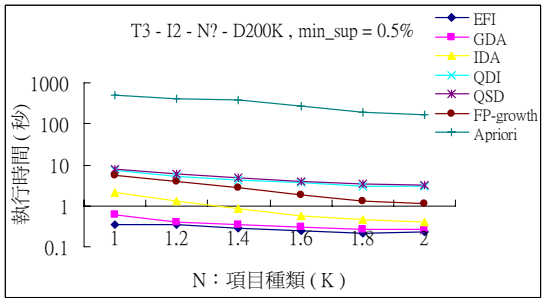

圖 30：在不同項目種類數量下，對各演算法效能的影響圖

由圖 30 可清楚得知當項目種類愈大時，每個演算法探勘所花費的時間就跟著變少，主要原因在於當項目種類呈現較多樣化時，每筆交易中可能會出現的商品種類也就愈多，即表示每個項目出現的次數也就相對的變少，因此，在相同門檻值的情況下，當項目種類愈多時，成為高頻項目集也就愈少，所以，每個演算法都是呈現出效能(特別注意的是，圖 Y 軸採用對數刻度)愈佳的情況；其中，EFI 演算法即使在項目種類較少的情況下，仍可避免產生大量不必要的項目集，所以，EFI 較其它演算法的執行效能佳且穩定。

(七)實驗七：在不同資料筆數下，對各演算法效能的影響

在實驗七裡，將比較在不同資料筆數且最小支持度門檻值為 0.5%下，對不同演算法效能的影響。同樣的，固定其它參數不變(T10-I5-N10K)，僅變動參數 D 的設定值，實驗結果如圖 31 所示。由圖 31 可得知當資料筆數增多時，各演算法所花費的時間都是呈線性成長；其中，QDI 演算法與 QSD 演算法在 D1,000K 時，呈現暴增的情況，主要原因為其記憶體已呈現不足的狀態，導致作業系統須額外將一部份的記憶體暫存在於硬碟中，以挪出記憶體空間來繼續執行程式，因此，其浪費一些時間在進行 I/O 處理上。另

一方面，Apriori 演算法則是明顯較其它演算法要來得慢，所以，在進階測試時，本研究將排除 Apriori、QSD 與 QDI 等演算法。在進階測試部份，因除了變動參數 D 外，也將 minsup 調低，以測試在資料筆數多且支持度低時，各演算法的執行效能變化，因此，IDA 也將予以排除，因為在支持度低較低時，IDA 必須拆解出完全項目集將暴增，也將導致記憶體無法負荷的情況發生，所以，進階測試部份僅進行比較 EFI、GDA 及 FP-growth 等演算法。

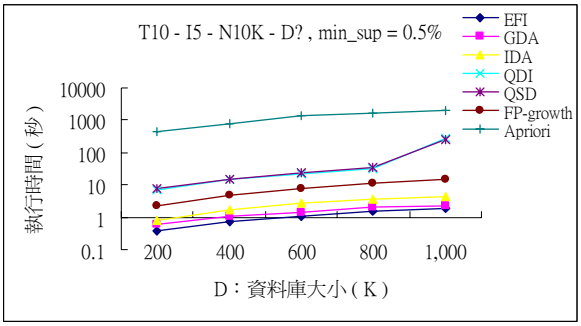

圖 31：在不同資料筆數下，對各演算法效能的影響圖

在進階實驗裡，產生六個資料量較大的資料庫，測試在這六個資料庫與最小支持度門檻值為 0.2%下，對不同演算法的效能影響。同樣的，固定其它參數不變(T10-I5-N10K)，僅變動參數 D 的設定值，實驗結果如圖 32 所示。

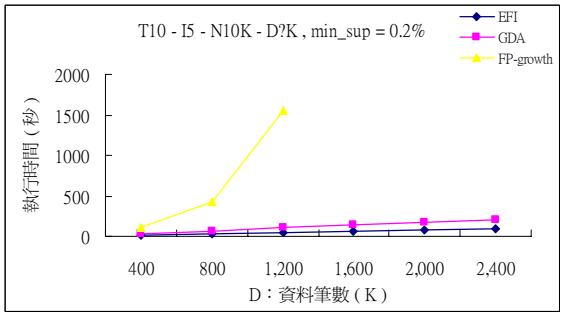

圖 32：在不同資料筆數下，對各演算法效能的影響圖(進階實驗)

在圖 32 可得知當資料筆數愈多時，FP-growth 演算法的建樹成本也就愈大，甚至發生記憶體不足而無法繼續執行的情況。GDA 演算法雖以階段性拆解方式來產生高頻項目集，但因其須不斷地重覆拆解，所以，導致其執行效能降低。EFI 演算法因本身的過濾機制，即使資料筆數較多時，仍可有效過濾大量非必要的項目集數量，因此，在執行效能方面，EFI 演算法較其它演算法要來得優且佳。

(八)實驗八：在 FoodMart 不同資料筆數下，對各演算法效能的影響

在實驗八裡，將使用 FoodMart(微軟公司，2000)的 sales_fact_1998_2 資料庫進行測試，其商品項目種類有 1,560 種，經過彙整後，交易記錄有 2,689 筆。本研究將隨機抽取

5,000、10,000、15,000、20,000 及 25,000 筆交易進行模擬實驗，由於 FoodMart 資料庫的項目及筆數較少且經過隨機抽取後，其資料的緊密性將會較高，所以，將排除在前幾項實驗裡，效能較不佳 Apriori 與 ICI-like 等演算法，僅將 EFI、GDA 及 FP-growth 等演算法列入本實驗部份，比較在不同資料筆數且最小支持度門檻值為 0.2%下，對 EFI、GDA 及 FP-growth 演算法效能的影響，實驗結果如圖 33 所示。由圖 33 可得知因交易記錄有 2,689 筆且由電腦隨機抽取，因此交易記錄的重複性是相當高的。但是 D 為 20K 時，可能因資料重複性不高，導致高頻項目數量減少，使得 GDA 能先過濾大部份非高頻的項目，來有效縮減交易記錄的長度，所以，GDA 在此資料集的效能要比在其它資料集時來得佳；然而，當潛在高頻項目集長度較長時，GDA 需拆解的階層也就愈多，因此所須耗費的時間成本也就愈多。FP-growth 演算法因適用於緊密性高的資料庫，所以，其執行效能要比 GDA 來得佳。在此次實驗裡，本研究僅是亂數抽取 FoodMart 資料庫裡的交易記錄，因此，在經過刪除非高頻項目後，將同時進行相同交易資料合併動作，以避免將執行效能耗費在拆解相同交易記錄上；另一方面，EFI 演算法因本身的特殊過濾機制，可避免產生大量非必要項目集且提高記憶體使用率，因此，EFI 的執行效能仍較其它演算法穩定且佳。

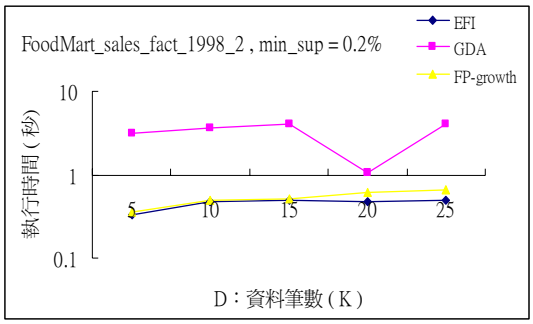

圖 33：在 FoodMart 不同資料筆數下，對各演算法效能的影響圖

在進階實驗裡，仍隨機抽取 FoodMart 的 sales_fact_1998_2 資料記錄，同時，也限定交易長度須為包含 10 或 10 以上，如此將可提高資料緊密性。本研究產生 5,000、10,000、15,000、20,000 及 25,000 筆等資料庫，測試各演算法在不同資料筆數且最小支持度門檻值為 0.275%下的執行效能，實驗結果如圖 34 所示。由圖 34 可得知，因交易記錄有 2,689 筆且由電腦隨機抽取，因此交易記錄的重複性是相當高的。但是 D 為 10K 時，可能因資料重複性不高，使得 GDA 能先過濾大部份非高頻項目，所以，GDA 在此資料集的效能要比在其它資料集時來得佳；然而，即使 GDA 採用階段拆解方式，卻仍無法在資料緊密性較高時，有效地提昇探勘效能，主要原因在於當高頻項目集長度較長時，將導致其須不斷掃描資料庫及產生大量的項目集，使得執行效能呈現不佳的狀態。FP-growth 演算法則因本身的資料結構較適合緊密性高的資料，所以，其執行效能是相當好。本研究的 EFI 演算法雖為拆解演算法，但因透過交易長度縮短及合併處理後，再經由特殊過濾機制產生新項目集等執行程序，即使在緊密性較高的資料庫裡，仍可與 FP-growth 擁有

同樣不錯的執行效能，所以，EFI 在資料緊密性較高的資料庫中，亦能以有效率的執行效能進行探勘。

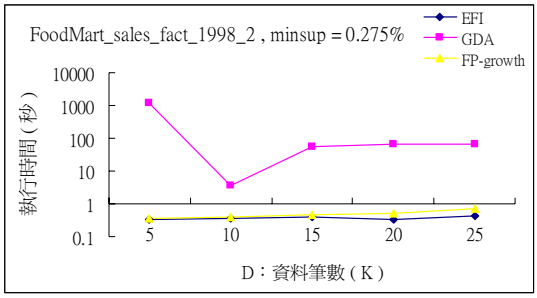

圖 34：在 FoodMart 不同資料筆數下，對各演算法效能的影響圖(進階實驗)

(九)實驗九：在大型資料庫下，對 EFI 及 Apriori 演算法效能的影響

在實驗九裡，將測試當大型資料庫無法一次讀入記憶體時，即表示資料庫容量大於記憶體容量時，對 Apriori 與 EFI 演算法執行效能的影響。本實驗使用 IBM Data Generator 產生出 5,000K、10,000K、15,000K、20,000K 及 25,000K 等不同資料筆數，並固定其它參數為 T10-I5-N10K。本研究將採用較簡單的分割方式，將大型資料庫分割為數個適當大小的子資料庫，以避免因大型資料庫無法一次讀入記憶體而中斷執行探勘任務。測試在不同資料筆數且最小支持度門檻值為 0.5%下，對 EFI 及 Apriori 演算法執行效能的影響，實驗結果如圖 35 所示。由圖 35 可清楚得知本研究所提出的新演算法 EFI，即使在記憶體受限的情況下，仍較 Apriori 演算法穩定及有效執行探勘程序，主要原因為 EFI 對於每個分割的子資料庫皆僅需進行四次 I/O 動作，也就是第一次找尋高頻 1-項目集、第二次刪除非高頻項目、第三次找尋高頻 2-項目集及第四次直接進行探勘產生新項目集等；另一方面，EFI 演算法不同於 Apriori 演算法會隨高頻項目集長度增長而增加 I/O 次數，EFI 演算法可避免因執行過多 I/O 動作，而浪費非必要的時間成本及可提高記憶體的利用率。

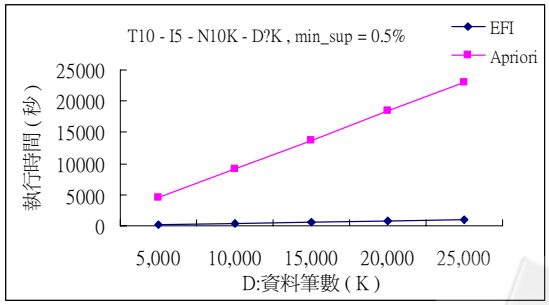

圖 35：在大型資料庫下，對 EFI 及 Apriori 演算法效能的影響圖

陸、結論

本研究提出一新關聯規則之演算法 EFI(An Efficient Approach for Filtering Infrequent Itemsets)，其主要目的為提升探勘高頻項目集的效能及提高記憶體的利用率。EFI 演算法僅需掃描資料庫四次，即可完成探勘任務。在執行程序中，EFI 演算法能藉由二階段過濾機制來避免產生大量非必要的項目集，以提高執行效能及記憶體利用率；且當記憶體無法一次載入大型資料庫時，亦可採用分割大型資料庫為 m 個可讀入記憶體的子資料庫方式，而每個子資料庫也僅需進行四次 I/O 動作，即可完成探勘任務因此。總合來說，EFI 演算法有以下的優點：

(1) 僅需掃描資料庫四次；若記憶體無法載入大型資料庫時，可採取資料庫分割成數個可讀入至記憶體的子資料庫方式，若有 m 個子資料庫，僅需作 4*m 次 I/O 動作，EFI 演算法不會隨著高頻項目集的長度增加 I/O 次數，EFI 演算法可避免執行過多的 I/O 動作，而影響執行效能。

(2) 在執行過程中，不用產生候選項目集，藉由過濾機制來減少大量非高繁項目集的產生，以避免時間浪費在產生非必要項目集上，亦可提高記憶體的使用率。

(3) 整個架構簡單且易實作，當記憶體充足時，執行效能將會更好。

未來可以再更進一步的研究方向為：

(1) 將項目表儲存的資訊更進一步壓縮以節省記憶體空間。

(2) 加入平行處理的能力，使效能再提昇。

(3) 應用在不同的領域，例如可探勘序列型樣(sequential pattern) 或週期性型樣(periodic pattern)的演算法。

參考文獻

1. 黃仁鵬、陳秀如(民 93)，高效率之關聯法則規則探勘演算法 QSD，南台科技大學資訊管理研究所碩士論文。
2. 黃仁鵬、黃南傑(民 93)，高效率拆解之關聯規則探勘，南台科技大學資訊管理系碩士論文。
3. 黃仁鵬、熊浩志(民 94)，快速資料探勘演算法與相關應用，南台科技大學資訊管理研究所碩士論文。
4. 黃仁鵬、熊浩志、郭煌政(民 93)，「直覺拆解之關聯法則演算法-IDA」，第十屆資訊管理暨實務研討會，國立台中技術學院資訊管理系暨資訊科技與應用研究所主辦。頁 1857-1866。
5. 黃仁鵬、錢依佩、吳聲弘(民 92)，「高效率之關聯規則探勘演算法－ICI，第十四屆國際資訊管理學術研討會，頁 155。
6. 微軟公司(2000)，Microsoft Analysis services 的範本倉儲資料庫 FoodMart。

7. R. Agrawal and R. Srikant, “Fast Algorithm for Mining Association Rules in Large Databases”, Proc. 1994 Int'l Conf. VLDB, Santiago, Chile, September 1994, pp.487-499.
8. S Brin, R. Motwani, J.D. Ullman, and S. Tsur (1997), “Dynamic Itemset Counting and Implication Rules for Market Basket Data”, ACM SIGMOD Conference on Management of Data, pp.255-264.
9. J. Han, J. Pei, & Y. Yin(2000), “Mining Frequent Patterns without Candidate Generation”, Proc. ACM SIGMOD Int. Conf. on Management of Data, pp.1-12.
10. J. S. Park, M. S. Chen, and P. S. Yu(1995),”An Effective Hash-based Algorithm for Mining Association Rules”, Proc. Of the ACM SIGMOD Conference on Management of Data, pp.175-186.
11. F. C. Tseng, & C. C. Hsu(2001), “Creating frequent patterns with the frequent pattern list”, Proc. Of the Asia Pacific Conference of Data Mining and Knowledge Discovery, pp.376-386.