南　華　大　學

資訊管理學系

碩士論文

連續長度編碼技術於資料流關聯法則之應用

Run length encoding -- based algorithm for mining association rules in data stream

研 究 生：陳宏隆

指導教授：邱宏彬

中華民國 九十七 年 六 月 二十 日

南　華　大　學

資訊管理學系

碩 士 學 位 論 文

連續長度編碼技術於資料流關聯法則之應用

研究生：（陳宏憲）

經考試合格特此證明

口試委員：

指導教授：

系主任(所長)：

口試日期：中華民國　97　年　6　月　27　日

誌　謝

本篇論文能夠順利完成，首先要感謝我的指導教授邱宏彬老師，從資料探勘的基礎概念、研究方向的制定、演算法的產生與修正、實驗步驟的指導、到最後手稿的逐字修正，老師都給予莫大的協助，才得以在兩年的時間內完成整篇論文。在這兩年中除了學到了很多新的知識外，老師的研究態度和對學生的多方照顧, 更讓我受益良多。

時光飛逝，兩年快樂的研究生涯已經結束，回首這兩年的生活點滴，學習到許多治學研究。本論文得以順利完成，要感謝許多人從旁的指導與協助。在此，願藉本論文一隅，表達心中無限的感激。感激之情，筆墨難以形容，僅此致上最誠摯的感激與謝忱。此外，最後要感謝我的父母、岳父母與妻子，因為有你們的支持與關懷，使我無後顧之憂，才能順利完成碩士學業，僅以本論文獻給我的家人、朋友、指導老師、同學，願與他們分享這份喜悅與榮耀。

陳宏隆 謹識
南華大學資訊管理學研究所
九十七年七月

連續長度編碼技術於資料流關聯法則之應用

學生：陳宏隆　　指導教授：邱宏彬

南　華　大　學　資訊管理學系碩士班

摘　　要

資料串流探勘是一個新興的研究領域，而關聯規則演算法是資料探勘(Data Mining)中一項相當重要且實用的技術。關聯式法則最主要的目的就是在龐大的資料中，把一些資料項目的相關性找出來，主要的方法是搜尋資料庫找出所有的高頻項目組；並且利用高頻項目組挖掘出所有的關聯式法則。因為大量的資料及大量候選資料項目組的產生，所以找出所有的高頻項目組是一件必須耗費大量計算成本的工作。因此如何有效的減少在計算時的資料量，減少候選資料項目組的產生，並且減少資料庫的讀取次數等，都能夠使關聯式法則挖掘的演算法更有效率。

我們研究的主要內容利用建構規則編碼(Run-Length Encoding)方式在動態資料庫中有效的減少關聯式法則演算法在計算時的資料量，主要的貢獻是提供一種新的資料前處理的方法，其利用將交易資料庫編碼成少量

的資料，然後直接對主記憶體中的編碼資料進行資料探勘，並且在快速資料異動時能夠有效更新編碼資料，加速演算法的執行速度，提升處理效能 。

關鍵詞：關聯式法則、資料挖掘、高頻項目組、候選資料項目組、支持度、變動資料、資料串流

Run length encoding –based algorithm for mining association rules in data stream

Student： Hung-Lung Chen Advisors：Hung-Pin Chiu

Department of Information Management
The M.I.M. Program
Nan-Hua University

ABSTRACT

It is a new developing research field that the materials bunch flows and prospects, and the RLEated rule performs algorithms and is prospected by the materials (Data Mining) A quite important and practical technology in China. The RLEated type rule main purpose is to find out the dependence of some materials projects in the huge materials. The main method is to search the database and find out all high-frequency project teams; And utilize the high-frequency project team to excavate out all RLEated type rules. Because of the production of a large number of materials and a large number of candidate materials project teams, it is that one must consume a large number of work of calculating the cost to find out all high-frequency project teams. So, what effective materials

amount while calculating of reduction, reduce the production of the project team of candidate materials, and reduce reading number of times,etc. of the database, the algorithm of performing that can make the RLEated type rule excavate is more efficient. Main content that we study utilizes and builds and constructs the regular code (Run-Length Encoding) The valid RLEated type rule of reduction in the dynamic database of the way performs the materials amount of algorithms while calculating, main contribution is the method of offering a kind of new materials to deal with, it utilized and traded the database and encoded a small amount of materials, then prospect the materials to the code materials mainly in the storing device directly, and can upgrade and encode the materials effectively in the unusual fluctuation of fast materials, perform the speed of execution of the algorithm with higher speed, improve and deal with efficiency.

Key word：Data Stream,Association Rules,Data Mining,
Change materials

目　　錄

圖 表 目 錄

第一章　　序　　論

現今是一個資訊爆炸的時代，每天所產生的資料量，可能已經超過以前累積的歷史資料總和。隨著各種組織機構的全面電腦化，再加上網際網路的蓬勃發展，資料的產生及流通皆呈指數地成長。如何有效地從這些龐大的資料量中，快速找出有用的資訊並且加以利用。已成為現在重要的課題。資料挖掘(data mining)最主要的目的，便是要從龐大的歷史資料當中，挖掘出隱含且有用的資訊，因而可做為決策時之參考，以幫助資料管理及制定決策，其中關聯法則是廣泛被應用以找尋資料中的項目或屬性之間共同發生的關係，例如找出商場中某些產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，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作改良，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一併作促銷。所以利用關聯式法則挖掘的目的，便是希望能從資料相對發生次數的分析著手，找出原本所未知的關係，作為決策時的參考。

傳統的資料庫與資料探勘方式已經無法達到使用者的要求，故產生了資料串流的概念；資料串流擁有資料異動頻繁、資料流入量龐大與使用者需要快速回應等特性例如網路流量分析(network traffic analysis)、網頁點選串流探勘(Web click stream mining)、網路入侵偵測(network intrusiondetection)以及線上交易分析(on-line transaction analysis)等，我們所要處理的資料不再是靜態的資料，而是一連串即時且連續的動態

資料流(dynamic data stream)；因此在這樣資料不間斷的異動下傳統探勘方法是無法達到使用者的要求，傳統的關聯挖掘如 ：Apriori因必須掃描多次資料庫導致效能不佳、Fp-tree雖不須產生侯選集，但因資料庫大時則會佔用較大的記憶體空間而且也不適合在漸進式挖掘(incremental mining)[1][2][3]或是在線上挖掘(Online mining)[4][5]、Bit-Vector的方式須佔用大量的記體，所以以上三種方法並不適合在Data Stream的環境中作探勘。在資料流環境中挖掘大型項目集，最主要的問題就是如何在無法看到所有資料的情形下，持續不斷的探勘出完整的大型項目集，而不會有任何大型項目集的遺漏。因為資料流動迅速，不允許回頭再次勘查資料，所以在資料流環境中挖掘大型項目集的演算法，必須是一個持續不斷的分析挖掘；Lin et al. [6] 提出一個在移動視窗中挖掘大型項目集的方法。

因此本論文提出了一個有效的演算法針對在資料流的環境下連續探勘，以Apriori及Bit-Vector觀念研究出一個連續長度編碼之資料流關聯法則挖掘法的演算法[7]，主要是將原始資料轉換成可以儲存於記憶體的結構,而當資料流動之後我們僅需對儲存於記憶體上的資料結構做局部的更新動作，將流出的資料刪除，將流入的資料加入，而資料探勘方法可以直接在主記憶體上執行而無須對資料庫做重複的掃瞄。目的便是在對動態資料庫(Dynamic Database)作關聯式法則挖掘時，可以讓使用者線上調整門檻值，並在設定門檻值之後的有限時間內得到規則，而在資料不斷更動的情況之下，系統也能提供與資料庫相符合的規則。也就是說，不管使用者如何調整門檻值，都可以不必對整個資料庫重新作挖掘，便能得到挖掘的結果。

經過實驗證明本論文所提出的演算法的確優於目前常見的傳統演算法。

第二章　文獻探討

2.1 關聯規則

2.1.1 關聯規則定義

關聯規則(Association Rule)是資料探勘中主要的技術之一1993 年由Agrawal 等人所提出[8-10]，通常又稱購物籃分析(Market-Basket Analysis,MBA)，主要是在大型交易資料庫中擷取項目(Item)之間的關聯性。關聯法則主要的問題定義如下：令$I=\{i_1, i_2, \cdots, i_m\}$，I 為所有商品項目(Items)的集合，D 為資料庫中所有交易記錄的集合，T 為每筆交易記錄項目的集合，T 為I 的子集合($T \subseteq I$)，而T 中的交易記錄內容是不考慮商品項目購買的數量，TID 為每一筆交易記錄的編號。

關聯法則以$X \rightarrow Y$ 表示，其中$X \subset I$，$Y \subset I$ 且$X \cap Y = \emptyset$。然而，一個關聯法則衡量的規則是什麼？如何才能讓一個關聯法則成立？衡量關聯法則的標準有二，支持度(support)與信賴度(confidence)，其探勘關聯法則的工作主要可分為二個階段，(一)找出資料庫所有的高頻項目集(Frequent itemsets)，也就是找出所有滿足最小支持度的項目組，若一個項目組含有k個項目，則稱為k-項目組(k-itemsets)，若k-項目組滿足最小支持度，則稱為k-高頻項目集；(二)根據第一階段產生的高頻項目集產生關聯法則；例如BCE 為3-高頻項目組，且B、C、E $\subseteq$ I，若關聯法則$BC \rightarrow E$ 滿足最小信賴度，表示此關聯法則成立。

而所謂關聯規則就是從大量的資料集合中，探勘在資料間具有相互關系

的隱藏知識。在關聯規則中非常著名的啤酒與尿布為例來說明什麼是關聯法則；假設每一個購物籃都代表一個單獨的交易，並且在每個籃子中的物品並不完全相同，而所謂關聯規則主要就是在這些資料裡面，找出其中是否具有相互關係，例如 A 購物籃裡有啤酒、尿布、報紙、牛奶，B 購物籃中有口香糖、啤酒、尿布，C 購物籃有啤酒、尿布、雞蛋；我們可以發現只要有購買啤酒的交易，消費者同時也會購買尿布，很有可能是太太要求先生去幫小孩購買尿布，而先生在同時也會買些啤酒來喝，經過分析之後，發現消費者具有如此的消費習慣時，則可以進行相關的決策。例如：顧客買了啤酒之後，其中 25%會同時購買尿布所把可以啤酒和尿布放在一起，以方便消費者的選購。在這樣的例子中我們可以很快的找出其中存在的關係，這就叫做關聯規則。

2.1.2 關聯規則探勘法目的

在一個資料庫 D 中，尋找出能夠同時滿足使用者之前所定義的最小信心值（Minconf）和最小支持度（Minsup）的所有 關聯規則。

2.1.3 關聯規則探勘方法

關聯規則之探勘過程主要包含兩個階段：第一階段必須先從資料集合中找出所有的高頻項目組(Frequent Itemsets)，第二階段再由這些高頻項目組中產生關聯規則(Association Rules)。關聯規則探勘的第一階段必須從原始資料集合中，找出所有高頻項目組(Large Itemsets)。高頻的意思是

指某一項目組出現的頻率相對於所有記錄而言，必須達到某一水準。一項目組出現的頻率稱為支持度(Support)，以一個包含 A 與 B 兩個項目的 2-itemset 為例，我們可以經由公式(1)求得包含{A,B}項目組的支持度，若支持度大於等於所設定的最小支持度(Minimum Support)之門檻值時，則{A,B}稱為高頻項目組。一個滿足最小支持度的 k-itemset，則稱為高頻 k-項目組(Frequent k-itemset)，一般表示為 Large k 或 Frequent k。演算法並從 Large k 的項目組中再產生 Large k+1，直到無法再找到更長的高頻項目組為止。

Support(A,B) = (同時包含項目 A 與項目 B 的筆數) / 所有交易筆數…(1)

關聯規則探勘的第二階段是要產生關聯規則(Association Rules)。從高頻項目組產生關聯規則，是利用前一步驟的 Frequent k-itemset (k≧2)來產生規則，在最小信賴度(Minimum Confidence)的條件門檻下，若一規則所求得的信賴度滿足最小信賴度，稱此規則為關聯規則。例如：經由 Frequent 2-itemset {A,B}所產生的規則 A→B，其信賴度可經由公式(2)求得，若信賴度大於等於最小信賴度，則稱 A→B 為關聯規則。

Confidence(A→B)
=(同時包含項目 A 與項目 B 的筆數) / (所有包含 A 的筆數)……(2)

因此，對於一個被表示為 A→B 的關聯規則，必須同時滿足下列兩個條件：Support(A,B) ≧ min_support 且 Confidence(A→B) ≧ min_confidence。

假設某賣場使用關聯規則探勘技術，對交易資料庫中的紀錄進行資料探勘，希望藉由過去多年所累積的大量交易紀錄，挖掘出潛藏在其中的商品組合關聯性。首先，我們必須設定最小支持度與最小信賴度兩個門檻值，在此假設最小支持度 min_support=5% 且最小信賴度 min_confidence=70%，因此符合此賣場需求的關聯規則將必須同時滿足以上兩個條件。若經過探勘過程所找到的關聯規則「香檳→冰塊」，滿足下列條件，將可接受「香檳→冰塊」的關聯規則。

Support(香檳,冰塊) ≧5% 且 Confidence(香檳→冰塊) ≧70%

Support(香檳,冰塊) ≧5% 於此應用範例中的意義為:在所有的交易紀錄資料中，至少有 5%的交易呈現香檳與冰塊這兩項商品被同時購買的交易行為。

Confidence(香檳→冰塊) ≧70% 於此應用範例中的意義為:在所有包含香檳的交易紀錄資料中，至少有 70%的交易會同時購買冰塊。

因此，今後若有某消費者出現購買香檳的行為，賣場將可推薦該消費者同時購買冰塊。這個商品推薦的行為則是根據「香檳→冰塊」關聯規則，因為就該賣場過去的交易紀錄而言，支持了「大部份購買香檳的交易，會同時購買冰塊」的消費行為。

2.2 資料探勘相關技術

2.2.1 Apriori 演算法

Apriori 演算法由Agrawal 等人(1994)所提出[10]，這個方法的主要概念是重複讀取資料庫，並且在每次讀取資料庫後產生長度相同的大項目集合（Large Itemsets）。另外只針對候選項目集合（Candidate Itemsets），而非所有可能的項目集合（Itemsets）來作支持度的計算，以減少計算時間來增進效率。主要包含以下步驟：

(1)利用(k-1)-高頻項目集(L_{k-1})來產生候選項目集合(C_k)。

(2)掃描資料庫D，計算所有候選項目集合的支持度，將所有支持度大於等於最小支持度的候選項目集合選出來成為長度為K的高頻項目集(L_k)。

(3)重複上面(1)(2)步驟，直到無法再產生新的候選項目集合為止。候選項目合併(Join)與修剪(Pruning)規則：

(1)由上述的步驟(1)找出有兩個k-2 項目相同的(k-1)-高頻項目集，組合成k-項目組。

(2)判斷(1)步驟中的k-項目組，其所有的(k-1)-項目組之子集合是否都出現，假如成立則保留此k-項目組。

我們在此用一個簡單的例子來說明Apriori 演算法的運作過程如圖2.1。資料庫中有六筆交易資料，假設最小支持個數為2。首先對資料庫中的六筆交易資料掃瞄一次，計算出每個項目的支持個數。可知有{A}、{B}、{C}、{E}的支持個數不小於2，即為大型1-項目集。接下來繼續進行第二階段，利用大型1-項目集產生長度為2 的候選項目集，會有{A,B}、{A,C}、{A,E}、{B,C}、{B,E}、{C,E}共六個候選2-項目集被產生，接著掃描資料庫計算其

支持個數，得到{A,B}、{A,E}、{B,E}三個大型2-項目集。第三階段依序產生候選3-項目集，並計算支持度，最後得到{A,B,E}大型3-項目集。

因為大型3-項目集只有一個，無法再產生候選4-項目集，所以Apriori 演算法探勘到此停止。

表1　交易資料範例

交易項目	編號
1	{A、B}
2	{A、B、E}
3	{B、D}
4	{C}
5	{A、B、C、E}
6	{B、E}

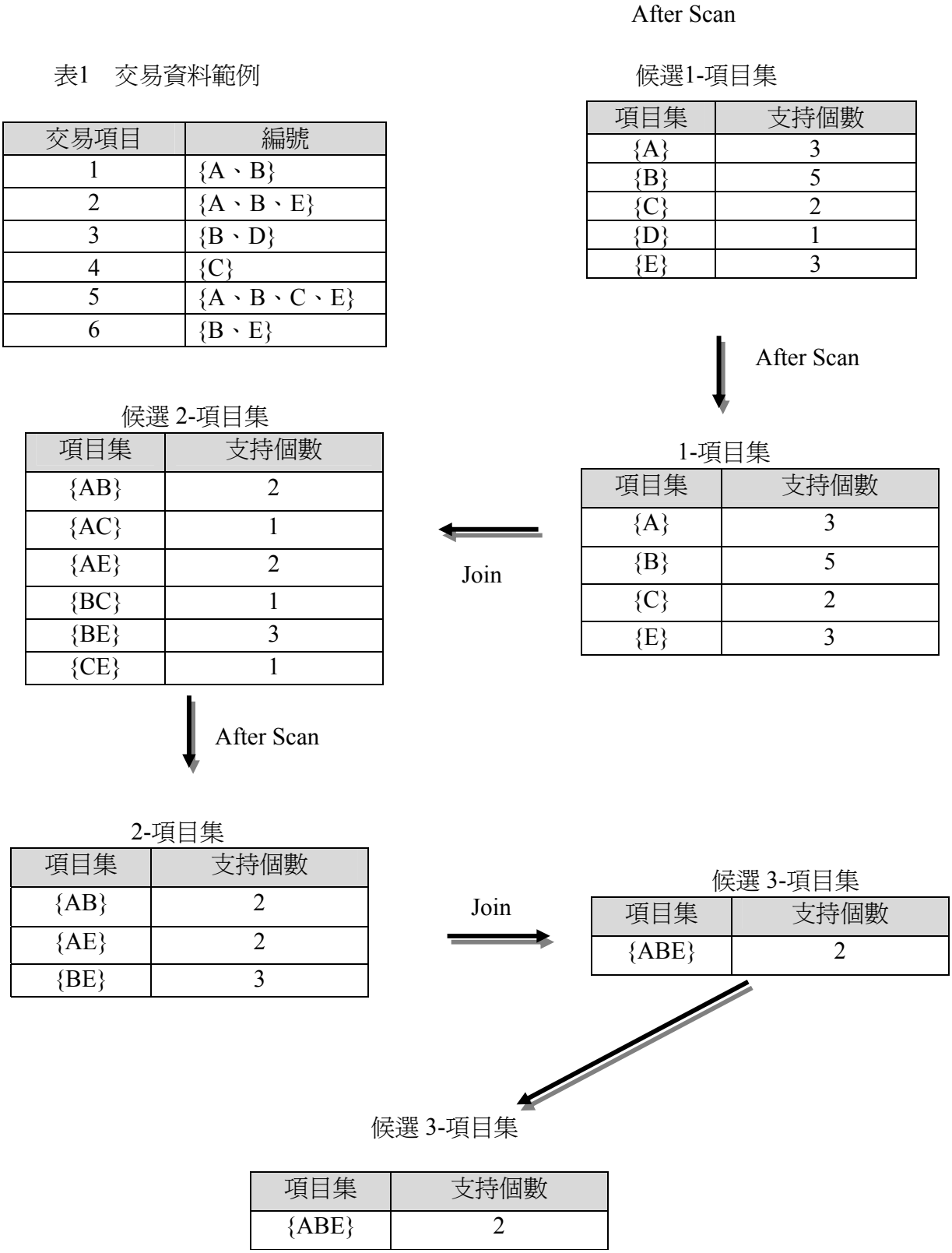


圖2.1 Apriori 演算法流程圖

```
1. L1={large 1-itemsets}
2. for (K = 2 ; LK-1 ≠ 0 ; K++) do begin
3. CK= apriori-gen (LK-1)
4. for all transactions    tI D do begin
5. Ct=subset (CK, t)
6. for all candidates c I Ct do
7. c.count ++;
8. end
9. LK={c I CK| c.count ≧ minsup}
10. end
11. Answer = ∪K LK;
```

圖2.2 Apriori 演算法虛擬碼

其中的 Apriori-gen 函式主要的功能就是產生候選項目集合，它利用 LK-1 來產生長度為 K 可能是大項目集合的所有項目。Apriori 的一個主要特色就是不斷地對磁碟進行讀取，來計算支持度，所以當資料庫的資料相當龐大時，此時就需要花費相當高比例的時間在磁碟讀取上，另一缺點是在於每一個回合會產生大量的 candidate itemsets，每一回合 candidate itemsets 產生後還需要再掃瞄資料庫來比對，而此步驟佔了整個演算法大部分時間，所以這是 Apriori 一個執行效率上的瓶頸。

2.2.2 Fp-Growth 演算法

FP-Growth演算法是由Han, Pei, and Yin（2000）提出[11]，此演算法

是不產生candidate itemsets作法的代表。它將資料庫壓縮在Frequent-Pattern tree的結構中，因為不用產生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。FP-Growth演算法的作法可分為兩個階段，主要步驟如下：

(一) 第一階段建立Frequent-Pattern tree：

(1) 第一次掃瞄資料庫,找出符合minimum support的large 1-itemset。

(2) 將每一筆記錄中large items依其出現在資料庫中的次數，作降冪排列。

(3) 第二次掃瞄資料庫時，建立FP- tree。

(二) 第二階段FP-Growth演算法搜尋FP-Tree產生高頻項目的步驟

(1) 對FP-Tree中的每一個分支node，建立conditional pattern base。

(2) 再對每一個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) conditional FP-Tree中有包含一條路徑，就可列舉出所有pattern。

FP-Growth演算法的優點為，不用產生候選項目集，而且將資料庫壓縮在FP-Tree的結構中，也改進了多次掃描資料庫的次數。

下我們用一個簡單的例子來說明FP-Growth 演算法的運作過程如表2.1~ 表2.5。

表 2.1 交易資料庫

TID	項目
T10	A,B,E
T20	A,D
T30	B,C
T40	A,B,D
T50	A,C
T60	B,C
T70	A,C
T80	A,B,C,E
T90	A,B,C

步驟一：掃瞄整個資料庫一次，找出每筆交易紀錄出現的次數，並列出來。

表 2.2 項目出現頻繁次數

Frequent Item	Support
A	6
B	7
C	6
D	2
E	2

步驟二：將每筆交易紀錄出現的次數，由大到小排列出來。

表 2.3 項目出現頻繁次數降冪排序

Frequent Item	Support
B	7
A	6
C	6
D	2
E	2

步驟三：把每筆交易紀錄的項目集調整好。

表 2.4 交易資料重新排序

TID	調整前項目	調整後項目
T10	A,B,E	B,A,E
T20	A,D	A,D
T30	B,C	B,C
T40	A,B,D	B,A,D
T50	A,C	A,C
T60	B,C	B,C
T70	A,C	A,C
T80	A,B,C,E	B,A,C,E
T90	A,B,C	B,A,C

步驟四：依照上述列出的項目集，描繪出一棵tree如圖2.3。

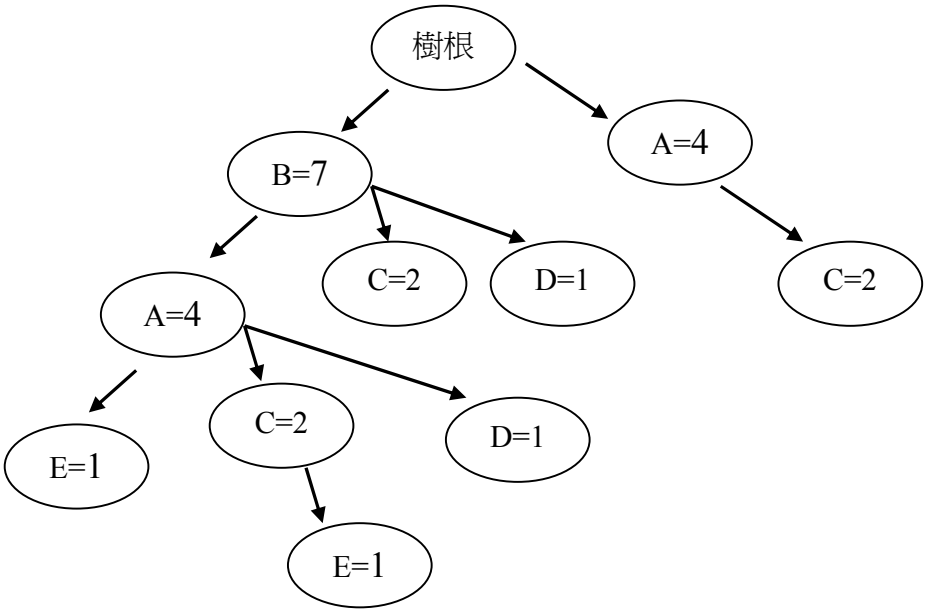


圖 2.3 Fp-tree

步驟五：再掃描資料庫一次，由下往上循序進行挖掘動作，刪除tree 中的子節點，就可以列出一個最終結果的表格如表2.5。

表 2.5 FP 演算結果

item	Conditional Pattern-Base	Conditional Fp-Tree	產生出高頻模式
E	{(B,A=1)、(B,A,C)=1}	{B=2,A=2}	BE=2 AE=2 BAE=2
D	{(B,A=1)、(B)=1}	{B=2}	BD=2
C	{(B,A=2)、(B)=2、(A)=2}	{B=4,A=2} {A=2}	BC=4 AC=4 BAC=2
B	{(A)=4}	{B=4}	
A	空集合	空集合	空集合

步驟6：根據上一個步驟表格，再依照Apriori 演算法的性質，則可以得到(B,A,E)、(B,D)和(B,A,C) 的子集合，也是經常的出現。

Fp-Growth之缺點：

在挖掘的過程中需要太多額外的處理時間及儲存空間去一直不斷地重覆產生大量的conditional base及conditional Fp-trees。

1. 在FP-Growth 演算法的探勘過程中，必須掃描資料庫兩次來建立FP-Tree，建置過程複雜且消耗許多時間。
2. 當資料庫很大或者使用者所訂的最小支持度很低時，建構FP-Tree 會消耗大量的記憶體。

2.2.3 DLG 演算法(Direct Large itemset Generation)

DLG 演算法[19] 主要在改進尋找高頻項目集的步驟，其概念是在產生高頻序列步驟中，只掃描資料庫一次並利用 Bit Vector 進行集合運算建立有向圖(Directed Graph)。利用有向圖指出項目間的關聯，並且追蹤有向圖產生 largeitemsets 及 sequential patterns。DLG使用Bit-Vector的紀錄方式來紀錄所有的交易資料，以減少掃描資料庫的次數。所謂的Bit-Vector，是將在資料庫中的每個項目種類，依據其在每筆交易中出現與否的情形，利用0與1來表示此項目在此筆交易中有出現，因此當交易資料庫中包含n筆交易紀錄時，則其每個項目的Bit-Vector的長度為n；故當資料庫中的資料量越大時，或是資料庫中的項變多時，DLG就需要更多的記憶體空間來儲存Bit-Vector。當建立完Bit-Vector後，再利用關聯圖來產生候選項目集，然後再對這對這些候選項目集的Bit-Vector進行AND運算以取得支持

度。

我們在此用一個簡單的例子來說明DGL演算法的運作過程如表2.6交易資料庫 ；我們假設最小支持度為2。

表 2.6 交易資料庫

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

利用 DLG 找尋所有高頻項目集，其步驟如下：

步驟一：首先對整個資料庫做一次掃瞄，將所有的項目紀錄下來，並計算每個項目的支持度，針對所有的項目建立 Bit-Map。所謂的 Bit-Map 就是記錄某個項目在每筆交易中是否有出現過的資料，當某個項目在某筆交易中有出現則將值設為「1」，若是沒有出現則將值設為「0」，所以當此資料庫有 n 筆交易時則每個項目的位元向量(Bit-Vector)的長度均等於 n(Bits)；如 A(1010)、B(0111)、C(1110)、D(1000)、E(0111)，故可知道 A 的支持度為 2、B 的支持度為 3、C 的支持度為 3、D 的支持度為 1、E 的支持度為 3，由以上可得 1-itemset 為 A、B、C、E。

步驟二：從第一個步驟中，任意選取兩個項目 Im 、In （m < n)，對這兩個項目的位元向量中的每個相對應的位元做計算，將結果中有多少個”1”，此數目就是代表這兩個項目在原始資料庫中同時出現的次數，計算支持度(Support)，將所有大於最小支持度 (Minsup) 的項目集合記錄下來，即

成 L_2。因此我們可得知 C_2 的候選項目集為 AB、AC、AE、BC、BE、CE，因此我們可以從 AND 運算後得到 L_2 高頻項滿集。

步驟三：建立關聯圖；根據 L，建立一個有向圖。假設項目 X 和項目 Y 是 L_2 中的一組項目集合，則在有向圖中就建立一條從 X 到 Y 的邊，每個邊都必須標上箭號，以便表示方向性如圖 2.4。

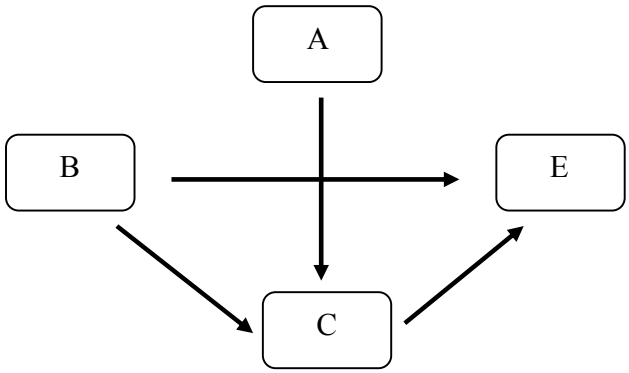


圖 2.4 DLG 關聯圖

步驟四：重複第二步驟到步驟三直到沒有新的候選項目集合產生為止

DLG 之缺點：

DLG 利用 Bit-Vector 的紀錄方式來紀錄交易資料，則每個項目的位元向量的長度，就是交易資料的筆數。因此，當交易資料庫中的交易筆數變多或交易資料庫中的項目變多時，DLG 就需要花費相當大的記憶體空間來儲存 Bit-Vector，當主記憶體的空間不敷使用時，其必須將部份資料存放於磁碟中，如此一來，將會降低演算法的執行速度，使得挖掘的效能並不能達到預期的效果。

2.2.4 Bit-Vector Data Format

所謂 Bit-Vectord[12]資料彙總(Data Summarization)是將在資料庫中的每個項目種類依據其在每筆交易中出現與否的情形，利用 0 與 1 來表示，其中 0 代表此項目在此筆交易中沒有出現，1 代表此項目在此筆交易中有出現。因此當交易資料庫中包含 n 筆交易時，則其每個項目的 Bit-Vector 的長度為 n；當資料庫中的資料量越大時，或是資料庫中的項目變多時就需要更多的記體空間來儲存 Bit-Vector 加以分析取其摘要。其方法可分為水平式與垂直式的資料切片方式。水平式是產生集合式的摘要而垂直式則是預測資料中欄位間的關係。此方式與其他預測模式最大不同之處在於他並非集中特定的欄位或群組間的預測，其終極目標是找出欄位間的關連性。此關連性的規則是在陳述某些組合發生的頻率與因果關係。

此分類主要是針對計算項目集支持度的方法，大部分的演算法都是採用水平式的資料配置，也就是交易資料庫內儲存的資料以 TID 為主，而交易的項目則是包含在每個 TID 中。另外也有些方法採取垂直的資料配置方式，也就是以記錄項目集為主，然後在每個項目集中記錄所出現的所有 TID 列表。

垂直的資料配置方式又稱為交集的方式，通常在項目中的 TID 列表都是以升冪的方式儲存，以提升整體的效率。

我們用 Bit-Vector 概念來檢視表 2.1 交易資料如下表 2.7。

表 2.7 Bit-Vector Format 檢視交易資料

Tid	Item	A	B	C	D	E
T10	ABE	1	1	0	0	1
T20	AD	1	0	0	1	0
T30	BC	0	1	1	0	0
T40	ABD	1	1	0	1	0
T50	AC	1	0	1	0	0
T60	BC	0	1	1	0	0
T70	AC	1	0	1	0	0
T80	ABCE	1	1	1	0	1
T90	ABC	1	1	1	0	0

如上表 2.7 第一筆交易紀錄 T10 包含項目為 ABE，故在交易的 Bit-Vector 為 11001。第二筆交易 T20 含項目為 AD, 故在交易的 Bit-Vector 為 10010。依序讀入交易，可以得到對應的各個項目的 Bit-Vector。

2.2.5 適用於Data Stream環境時間模式

2.2.5.1 Landmark model

指系統的起始時間，所以在標界模式中，關連法則探勘的問題相當於是從系統一開始一直到現在所產生的資料中發掘大型項目集如圖2.5。

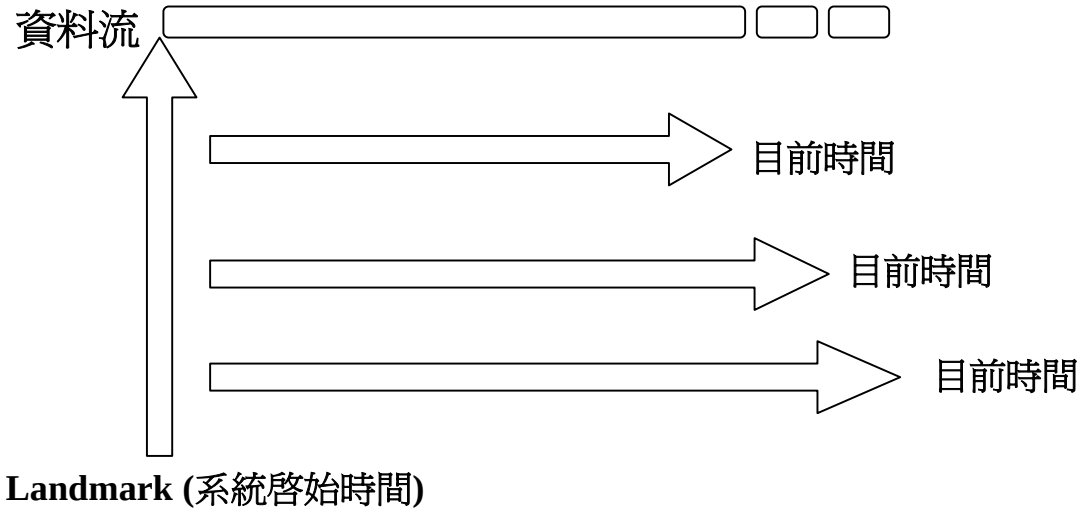


圖 2.5 Landmark model

2.2.5.2 Time-fading model

在Chang and Lee [13]所提出的Time-fading 模式中，對於資料給予適當的權重值，資料的權重值隨著時間的增加而遞減，相反的越新的資料能獲的越高的權重值，如圖2.6 所示。主要是增加時間和資料之間的關連性考量，改進在標界模式中未考慮到時間的重要性，因為我們相信越接近現在的時間點，資料的價值程度越高。

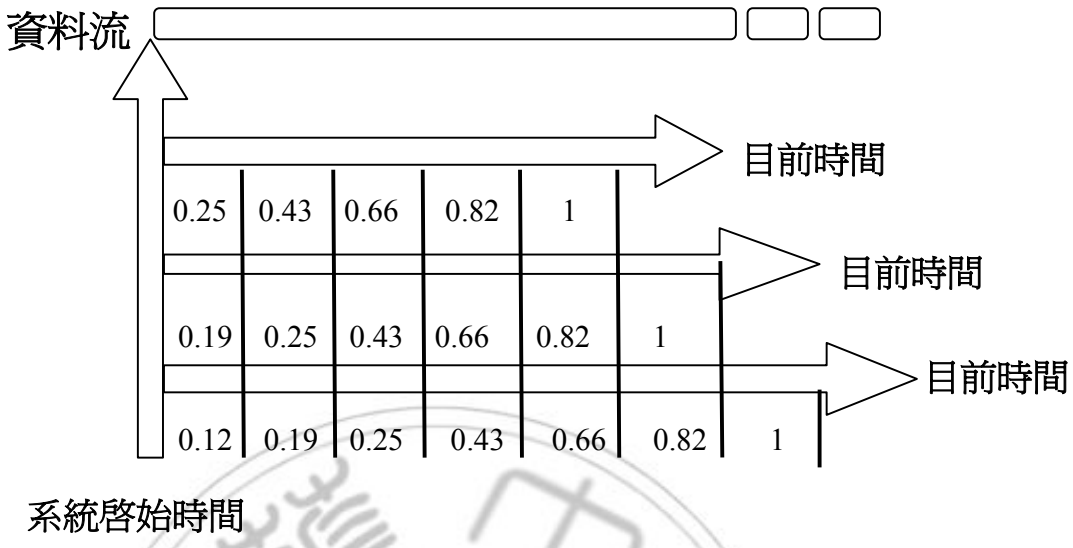


圖 2.6 Time-fading model

2.2.5.3 Sliding-Windows Model

Lin et al. [6] 提出一個在移動視窗中挖掘大型項目集的方法，移動視窗的大小是以時間來定義。他們設計一個大型項目集的儲存架構，是由兩個資料結構所組成: discounting table 儲存每個視窗中的大型項目集與其支持個數，而Potentially Frequent-itemset Pool 儲存在所有視窗和個別視窗中的大型項目集。他們提供兩種輸出模式，輸出所有精確和可能是的大型項目集。並且，針對在有限的記憶體中，設計一套調整discounting table的機制。當記憶體滿載時，合併discountingtable 中相同大型項目集在每個視窗中的支持個數，縮減記憶體的使用，讓新增資料能放入記憶體中。雖然允許某個程度上的誤差，但如果合併次數頻繁，輸出的精確性也隨之降低。

滑動視窗處理模型具有易於理解、設計簡單等優點，因此在數據流挖掘

中也得到廣泛的研究和應用 2003 年，Teng 等提出了一種稱為 FTP-DS 算法，它是在滑動視窗中使用統計回歸技術來挖掘頻繁項集 2004 年，Chi 等給出了 Moment 算法，它也是基於滑動視窗技術的，但是 Moment 僅關注在數據流中挖掘頻繁閉項集，因此它可以被期望來減少內存數據架構的規模和獲得較高的挖掘效率

這個方法設定了資料的單位數量，就某些資料量內的資料進行探勘。而因為新資料的增加而刪除了舊資料，類似平移的方式來取出範圍內的資料進行探勘。這樣不僅可以減少資料量不斷增加的問題，也可以達到忽略舊資料的參考性。在傳統移動視窗模式(sliding window model)的環境中[14] [15]，假設所給定的移動視窗之長度為T。關連法則探勘的目的是要從這T 筆交易中發掘所有的大型項目集。雖然移動視窗模式只考慮離現在最近的T 筆交易，移動視窗的大小是以時間來定義，而非以包含的交易個數來決定。以時間來定義的目的是為了避免在不同的時間點，相同數量的交易所涵蓋的時間範圍差異太大。

舉例來說，在圖2.7 中，假設視窗大小為3，在 Am 08:20 時，我們所要探勘的資料範圍為交易編號200、300、400，接著，在Am 10:20 為交易編號400、500、600，因為交易編號200、300已經不再移動視窗的範圍內。依序下去，則Am 13:00 的探勘範圍是交易編號500、600、700。

交易時間	交易編號	交易資料
Am0 7：00	200	A、B
	300	C
Am08：20	400	B、C
Am10：20	500	C、D
	600	A、C
Am13：00	700	G

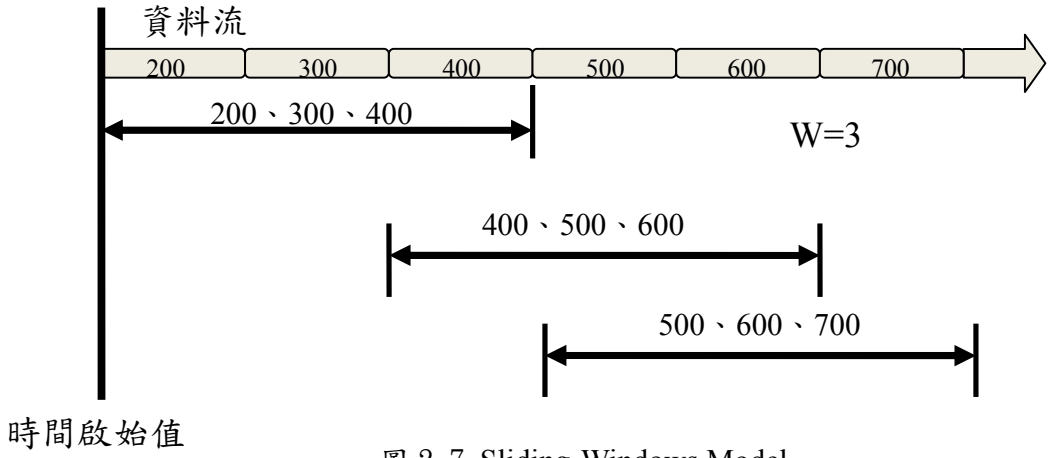


圖 2. 7 Sliding-Windows Model

第三章　研究方法

本研究針對Apriori 演算法、FP-Growth演算法及DLG演算法的缺點做改進如表3.1，因為在每一個產生頻繁項目的階段都必須掃描資料庫N次，相當的費時，也造成整體執行效率不佳。所以本研究的目的在於改善找尋關聯法則時所需花費的時間。藉由將交易紀錄轉換為以項目出現頻繁次數序列為主的交易變動長度編碼(Run-Length Encoding ,RLE)，來替代所有的交易紀錄。這份經由轉換過的編碼儲存所需空間將遠小於傳統靜態資料庫模式的儲存。使用者在使用這套編碼進行資料探勘時，只需要讀取編碼內容，而不需要對整個原始資料庫進行掃描。對於新加入的資料項目以及過時應要刪去的資料，我們只需對增加或減少的交易項目增減編碼長度，即可維護交易變動長度編碼並達到搜尋關聯規則的目的。在計算支持度時，也不需要將編碼作解壓縮的動作，即可求得正確的支持度。此外，由於交易變動長度編碼資料順序具有反應時間的特性，故相當適合用來處理在資料串流上的問題。

我們將對該演算法分為三個章節來說明，3.1 節我們對系統架構流程來說明，3.2節為編碼前置處理，我們將說明交易變動長度編碼的建構、新增、與刪除；3.3節為我們所提出的資料探勘演算法，說明如何利用編碼內容來進行資料探勘並求得支持度。

表 3.1 Apriori、Fp-Growth、DLG 演算法優缺點

演算法	优點	缺點
Apriori	利用Apriori 演算法來作關聯式法則挖掘，具有演算法容易實作、產生的規則容易應用的優點。	主要原因在於挖掘過程中必須多次掃描資料庫，且計算的項目繁重，並在挖掘完成後，一旦有門檻值或資料的更動都會造成必須重新作挖掘
FP-Growth	主要概念是不產生 candidate itemsets，而將資料庫壓縮在FP-tree 的結構中，以避免多次的高成本資料庫掃瞄。	建置過程複雜且消耗許多時間且當資料庫很大或者使用者所訂的最小支持度很低時，建構FP-Tree 會消耗大量的記憶體。
DLG	DLG 演算法主要的優點是產生 candidate itemsets 容易 ，且僅需掃瞄資料庫一次	DLG就需要花費相當大的記憶體空間來儲存Bit-Vector，將會降低演算法的執行速度，使得挖掘的效能並不能達到預期的效果。

3.1 資料探勘演算法系統架構說明

本文所提的關聯規則探勘方法方式與Apriori 類似。先產生候選項目集合，然後再計算這些候選項目集合的支持度，將符合條件的保留下來，不符合的刪除，一開始我們以Sliding-Window 模式先定義對Data Stream挖掘範圍，以W(Window)為探勘的框架， Bucket數為W的大小，如W=3即框架內(W)包含了3個Bucket，每次系統讀取1個bucket時，便會將框架尾端的bucket 刪除，以至框架不斷的位移以達到模擬資料串流的環境，不斷的對移動框架作探勘。整個系統流程如圖3.1所示。

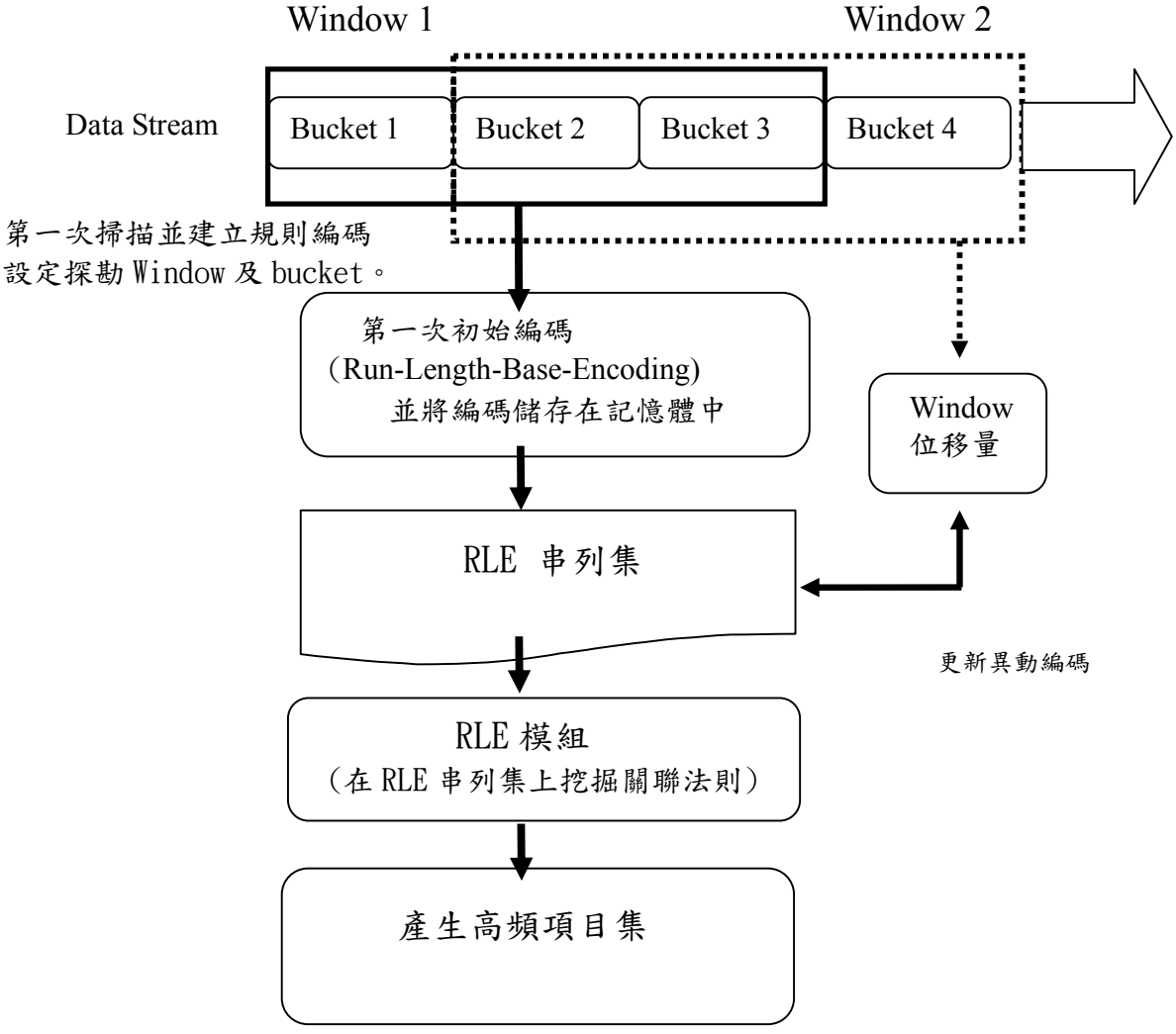


圖 3.1 系統架構流程圖

3.2 初始RLE串列集的建立

在資料串流的環境下，隨著時間資料量不斷的增加，資料快速的異動，以致於無法以靜態的探勘方式來對異動頻繁的資料庫做掃描的運算，因此我們提出一種編碼方式以記錄資料庫中的每一時段所輸入的資料，並以滑動視窗模式以不斷位移框架達到模擬資料串流。

模擬資料串流，如下圖3.2 假設欲探勘的資料量框架(Window) W=3, 表示整個框架的大小(bucket)在此我們可以看成3個時段(表示搜集3個連續時段的資料),所以如下一個時段(bucket)進入框架時，則原先在框架最早的時段(bucket)必須刪除，以讓最新時段的資料移入框架後進行探勘的工作，在這樣一進一出的資料位移模式下模擬成資料串湧流並同時對框架異動後的交易資料進行探勘。

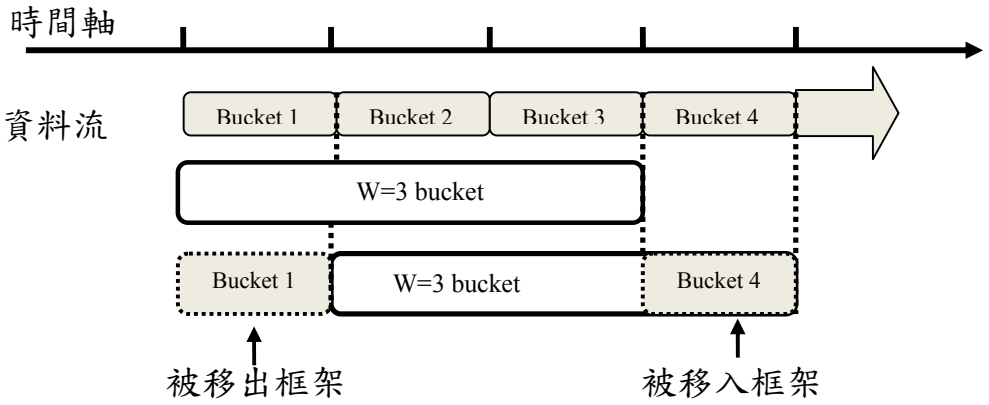


圖 3.2 資料流探勘示意圖

註：1.w=3 bucket

2.隨著時間的移動，框架內原本存在bucket 1、bucket 2、bucket3

3. bucket 1被移出框架，bucket 4被移入框架

4. 移動後則，產生新的框架 bucket 2、bucket 3、bucket 4

3.2.1 前置建構編碼

進行對資料庫交易資料探勘時,在最起始的時間點必須建構框架內的交易資料編碼，其建構編碼的定義如下：

定義一：項目(Item) I 之 Run Length Encoding(RLE)

假設項目I從t筆交易起，連續出現在交易

t,t+1,t+2,... ...,t+(**l**-1)之中，則稱交易出現項目I的一個Run，其長度為**l**；而其RLE表示為(t, **l**)。

定義二：項目(Item) I 之 交易RLE List

若項目I在交易中共出現k個Run，其RLE為

(t_1, $\mathbf{l}_1$), (t_2, $\mathbf{l}_2$),.. .. ., (t_k, $\mathbf{l}_k$)

則項目I的RLE List定義為

[(t_1, $\mathbf{l}_1$),(t_2, $\mathbf{l}_2$),.. .. ., (t_k, $\mathbf{l}_k$)]。

如下表3.2~表3.3是交易資料。

表3.2資料庫交易資料及Bucket資料

	交易ID	交易資料
第 1 個 bucket	T01	BCD
	T02	AF
	T03	BEFG
第 2 個 bucket	T04	ABE
	T05	AEF
	T06	ADEF
第 3 個 bucket	T07	BDE
	T08	BCDG
	T09	AB
第 4 個 bucket	T10	BCG
	T11	AEG
	T12	BG

W=3,假設每1bucket有3筆資料，即整個框架內的交易資料筆數為9筆，則我們分析到前9筆(1~9)交易資料將交易資料轉換成交易編碼如上表3.2

表3.3交易項目編碼

交易項目	每個Run的RLE
A	[(2,1),(4,3),(9,1)]
B	[(1,1),(3,2),(7,3)]
C	[(1,1),(8,1)]
D	[(1,1),(6,3)]
E	[(3,5)]
F	[(2,2),(5,2)]
G	[(3,1),(8,1)]

如上表3.3以交易項目A為例交易項目的位置及長度為(2,1)、(4,3)、(9,1) 如定義一;(2,1)則表示在第二筆交易出現A的項目有1次則長度為1；(4,3)則在第四筆交易開始第五筆及第六筆都有出現，則表示從第四筆資料開始連續三筆資料都有出現，長度為3，在第九筆又出現一次則長度為1。

因此A項目的 Run List為[(2,1),(4,3),(9,1)]如定義二， 此Run List即是交易編碼串列。

3.3 框架移動的 RLE 串列集更新

3.3.1 移出資料之編碼更新

當時間位移到下一個時段(即下一個bucket), 則我們必須把框架內最舊的資料從框架內刪除,也就是存在框架內最早的資料移除並對編碼更新如下圖3.4。其作法是依序處理RLE串列中的每個RLE。

定義三：假設項目I的RLE為(t_i, l_i),位移量為b

若(t_i-b)＞0, 則更新為(t_i-b, l_i)即新的index…… (1).

若(t_i-b)≦0,時

{ 若(t_i+ l)＜b ,則移除(t_i, l_i)……(2).

{ 若(t_i+ l)≧b ,則更新為(b-t_i+1, l_i-(b-t_i+1))… (3).

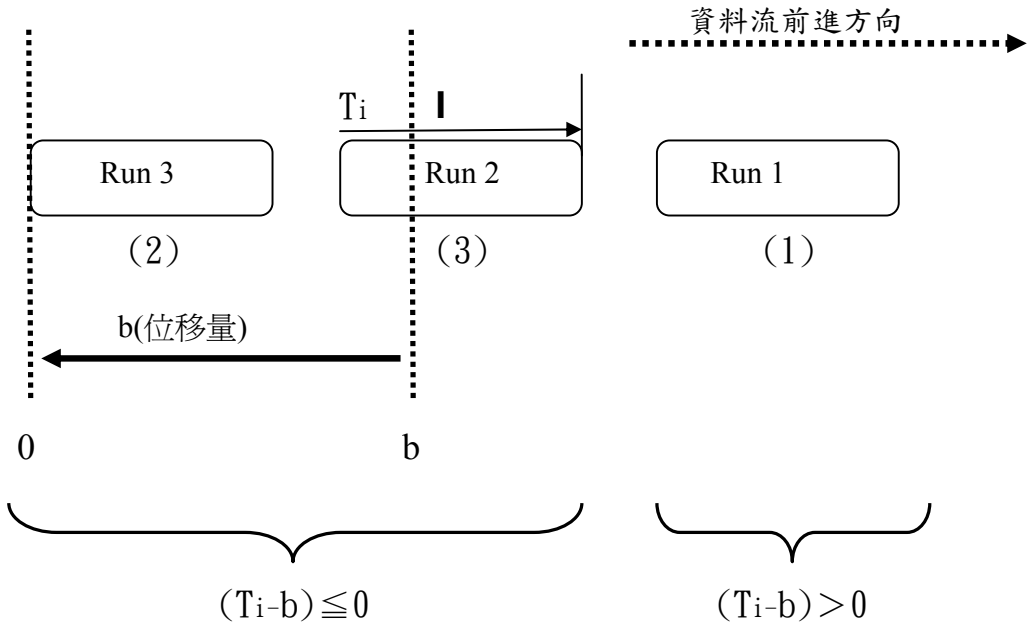


圖 3.4 移出資料之編碼更新

故在此我們將移除框架內最早期的資料以表3.2資料庫交易資料及Bucket資料來看為T01、T02、T03，故必須對此三筆中的交易項目編碼做移除。

表3.4更新bucket交易編碼

交易項目	每個Run的RLE
A	[(2,1),(4,3),(9,1)]
B	[(1,1),(3,2),(7,3)]
C	[(1,1),(8,1)]
D	[(1,1),(6,3)]
E	[(3,5)]
F	[(2,2),(5,2)]
G	[(3,1),(8,1)]

如上表3.4有標註的組別為第1個bucket資料，故令bucket長度為3以內的都會影響到，依序處理RLE串列中的每個RLE (T_i, l_i)，以A項目舉例說明：在第二筆交易資料出現一次(2,1)在長度3的範圍內，所以必須做處理，再往下判斷(4,3)是否在長度3的範圍內，如否，則接下來的資料就不可能影響到了，故把(2,1)從編碼中移除後再重新分配索引值(index), 產生新的編碼為[(1,3),(6,1)]如下圖3.5。

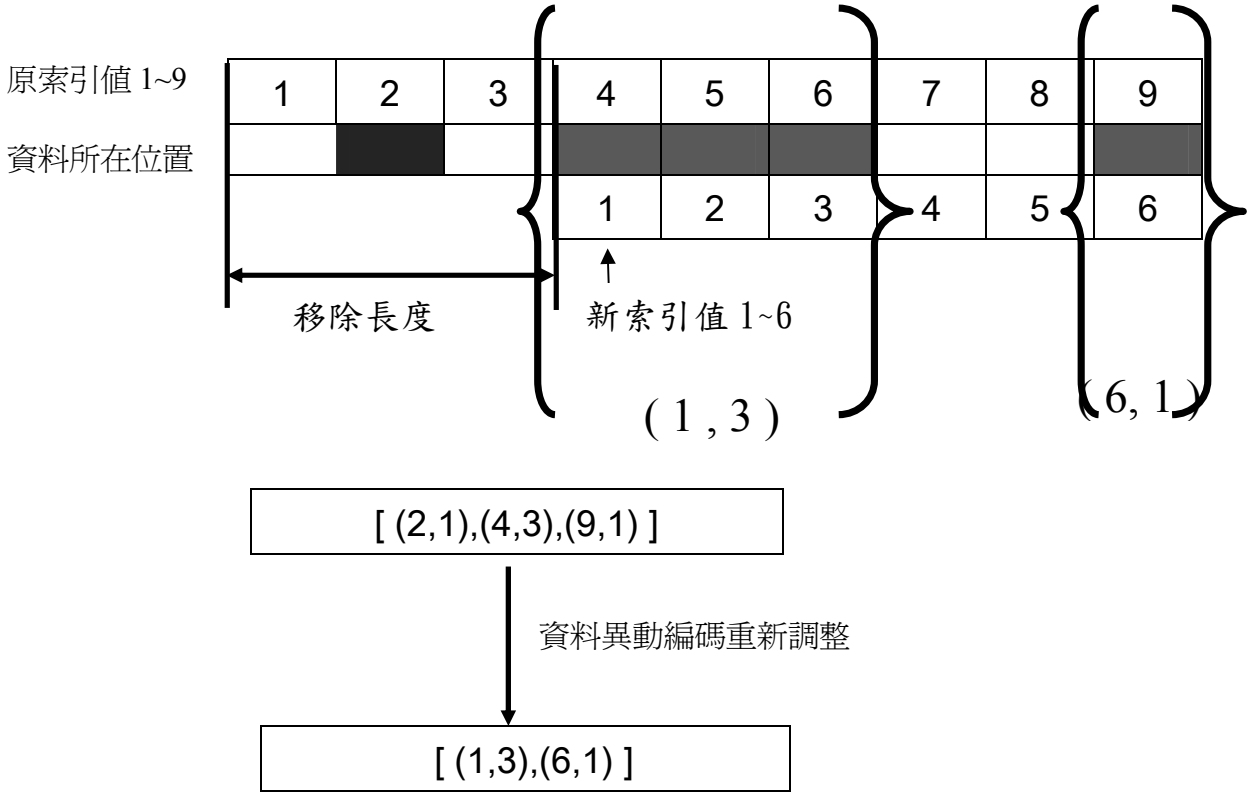


圖 3.5 重新產生編碼索引

位移後的編碼如下表3.5

表3.5刪除bucket交易編碼後結果

交易項目	交易編碼
A	[(1,3), (6,1)]
B	[(1,1), (4,3)]
C	[(5,1)]
D	[(3,3)]
E	[(1,4)]

F	([2,2])
G	[(5,1)]

3.3.2 新增編碼

對於最新流入的bucket資料，則將逐筆資料讀進來判斷，如新流入的資料item未存在於記憶體中時，則在記憶體中加入此新項目，否則直接在已存在於記憶體中的項目加入編碼。

定義四：假設目前已有k筆交易，目前處理的是新流入的bucket中的第i筆交易中的項目S。

(1)新增：若RLE 串列集中未出現項目S，則加入RLE

List[((k+i),1])]..... (4)

(2)附加與串接：若已有項目S的RLE List[(t_1, l_i),..,(t_m, l_m)]

(a)串接：若(t_m+ l_m)=(k+i)，則將(t_m, l_m)更新為(t_m, l_m+1)... (5)

(b)附加：若(t_m+ l_m)<(k+i)，則進行附加 新的RLE

[(t_1, l_1),..,(t_m, l_m),(k+ i,1)]。..... ....(6)

例如假設最新流入的資料是第4個bucket，則交易資料為T10~T12如下表3.6。

表 3.6 處理下一個 bucket 交易資料

交易ID	交易資料
T10	BCG
T11	AEG
T12	BG

編碼新增規則：

編碼附加情況：

步驟一：以交易T10交易資料項目B而言，在記憶體中啟始位置是7則表示B[(7,1)]。

步驟二：把T10的交易資料與已存在的資料做比對，如項目已存在且與該項目的是緊鄰的情況則編碼直接做附加(定義四-(6))。如B已存在編碼是[(1,1),(4,3)](表3.5)表示資料的最後位置是從4開始長度是3，所以位在記憶體的第4 到第6筆位置都出現B，故未值是6且新進的資料是從7開始長度為1，原始資料末值是6；新進資料啟始值是7，兩值相差為1表示緊鄰則直接對該項目原始編碼符加1(定義四-(6))，附加後的編碼為(4,4)如下圖3.6((4,4)表示從第4位置開始連續出現B的項目有4次，長度為4。

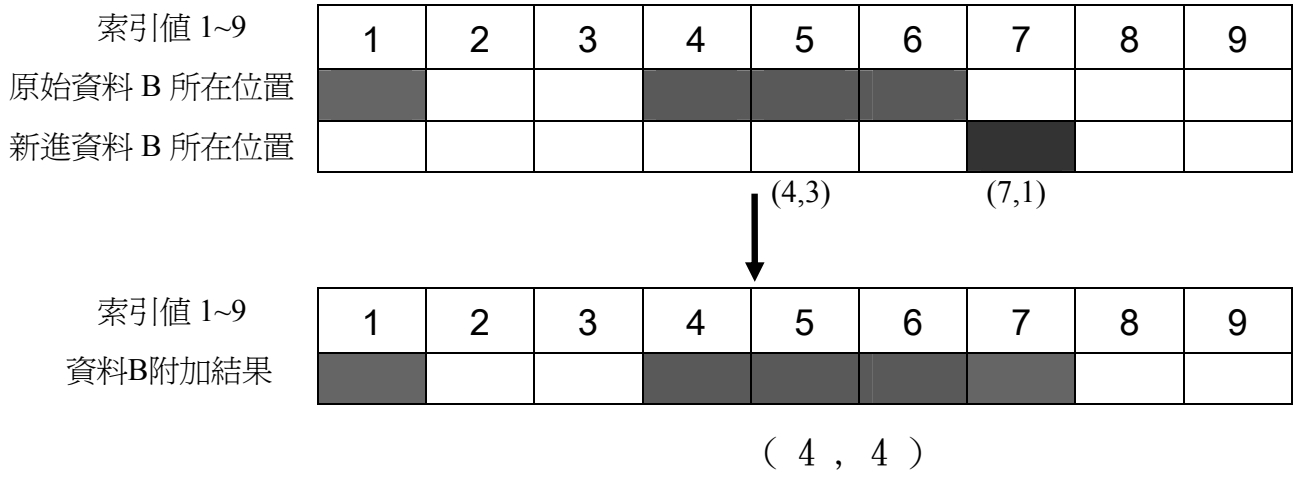


圖 3.6 新進資料規則編碼附加狀況

編碼串連情況：

步驟一：以交易T10交易資料項目C而言，交易啟始位置是7則表示置是7則表示C[(7,1)]

步驟二：把T10的交易資料項目C與已存在的資料做比對，因C的項目已存在編碼為[(5,1)](如表3.5)，新進資料以(7,1)表示則(5,1)與(7,1)中間誤差1不是緊鄰的情況則直接對編碼串連。如C已存在編碼是(5,1),則直接對編碼串連(7,1)，串連後的編碼為[(5,1),(7,1)]如圖3.7。

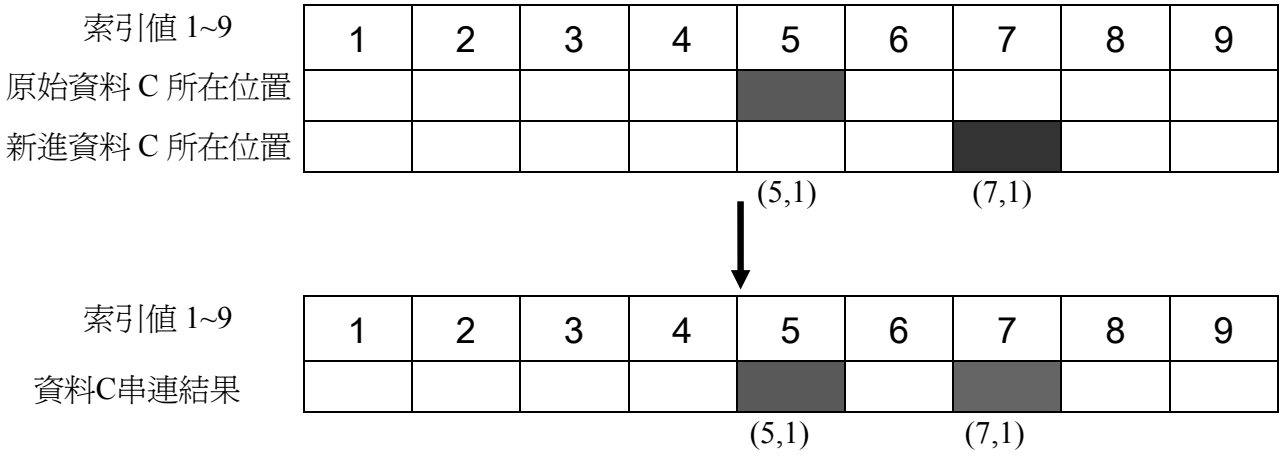


圖3.7新進資料規則編碼串連狀況

接著我們來檢視編碼(RLE)的內容代表意義，RLE 串列是由一串的連續正整數所組成，例項目 A 的 RLE 為[(2,1)、(4,3)、(9,1)]，表示存在記憶體中為(2,1)、(4,3)、(9,1) 如圖 3.5 所示，(2,1)表示從位置 2 開始出現 1 次，長度即為 1，故我們把記憶體中的每一組的第二個數目全部加總起來即為該項目 A 出現的總次數，加總後為 5，故在此的前置編碼即可尋找出

1-itemset，把 1-temset 當成候選集，經由 RLE 模組來產生 2-itemsset 或是 3-itemset，甚至更多高頻的項目集。

當我們對一個資料流做過資料探勘後，仍然會有新的資料進入資料庫中，因此經過一段時間之後，原有的關聯規則就無法正確地反映出資料流中新的狀況，此時就發生漸進式關聯規則更新的問題。最簡單的解決方法，就是對新的資料框架，重新執行一次資料探勘。不過資料探勘是一個相當耗時的工作，所以目前處理關聯規則更新的方法，均利用原有的關聯規則作為基礎(Run-Length-Base-Encoding)，來對新的資料流進行資料探勘以增進效率。

3.4 RLE 串列集之關聯法則挖掘

下圖3.8為RLE 模組架構說明

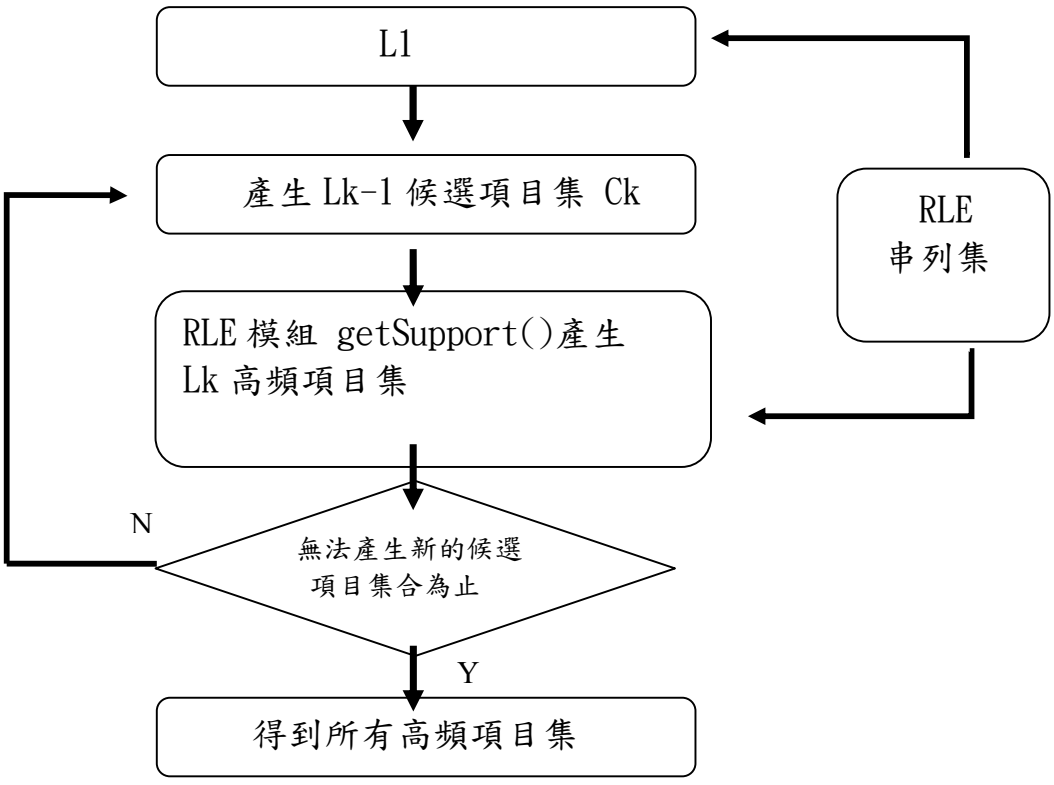


圖 3.8 RLE 模組架構說明

RLE比對產生高頻項目集步驟說明：

步驟一、將Lk-1的高頻重新組合產生Ck的候選集。

步驟二、將Ck候選集的RLE List 帶入RLE模組。

步驟三、And_op()判斷所屬分類狀況進行交集比對。

步驟四、GetSupport()回傳此次候選項目所比對的結果k頻繁項目集。

步驟五、以遞迴的方式重複步驟一到步驟四直到無法產生新的候選項目集合為止。

新探勘規則演算法如下圖3.9

```
1.  Run-Length-Base-Encoding
2.  L1=find_frequent_1-itemsets(D);
3.  for (K = 2 ; Lk-1 ≠ 0 ; k++) do begin
4.  {
5.     Lk=ψ;
6.     Ck-itemset = apriori-gen (Lk-1)
7.     for all candidates c in Ck-itemset do;
8.       {
9.            Count= getSupport(CK-itemset,win_sup)
10.           If(count≧min_sup)
11.           Lk=Lk∪{ C };
12.         }
13.   }
14.  Return L=∪kLk
```

圖 3.9 新探勘規則演算法

3.4.1 RLE各項交集狀況示意圖

當Data Stream中框架內的資料被壓縮編碼成RLE串列集而存在主記憶體內而進行關聯法則的挖掘。為了提升效率我們提出一個可直接在RLE串列集上進行挖掘的演算法如上圖3.9。經由上述的編碼規則之後，再以RLE模組來判斷兩兩項目的組合交集的狀況，來產生2項目的候選集，甚至尋找更高頻的項目集，如下圖3.10我們把所有可能項目交集的情況分為兩大類，第一類為A、B、C三種情形，第二類為D、E、F另三種情形，dest項目與Src比對情況可能有下列6種情況A、B、C、D、E、F，例如與dest沒有交集的情況為A或F，以下我們針對RLE模組來說明各個不同情況下的交集以計算交集的長度。

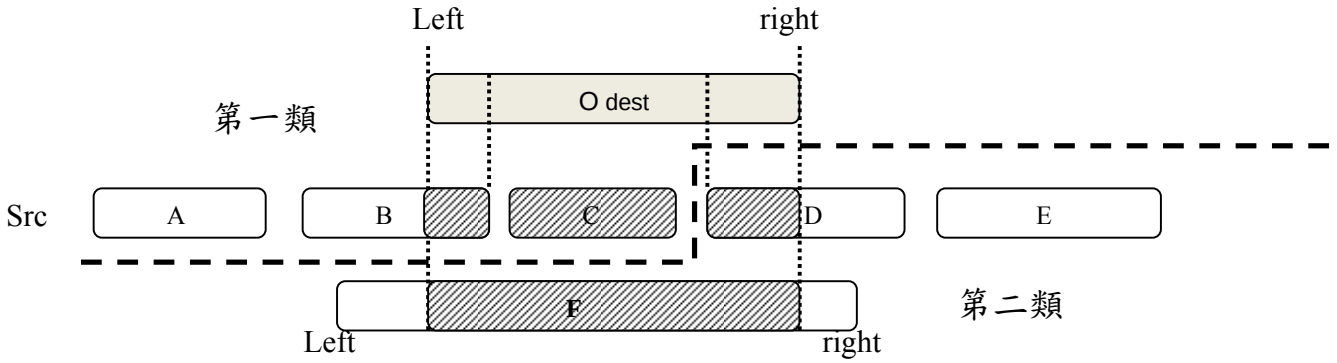


圖3.10 RLE模組架構情況圖

假設項目O之Run1為(dest), 項目P之Run1為(Src),一開始我們以dest_right與Src_right比較，如dest_right≧Src_right則屬於第一類包含了A、B、C三種情況，否則dest_right<Src_right則屬於第二類包含了D、E、F三種情況。

假設dest編碼為(3,4)：

第一類情況如下：dest_right≧Src_right

(A)沒有包含情況：當P為(1,1)則與dest比對都無交集的情況如下圖3.11。

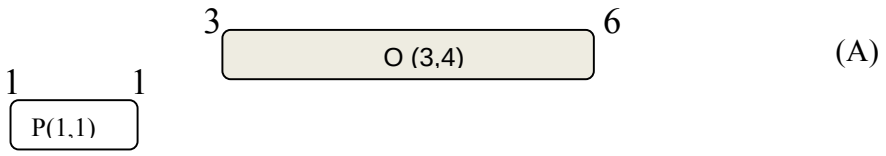


圖 3.11 RLE 模組 A 情況圖

(B)部分包含情況：當P為(2,4)則與O交集有2長度如下圖3.12所示，故OP的交集的部分從位置3開始到4，則以OP(3,2)表示之。

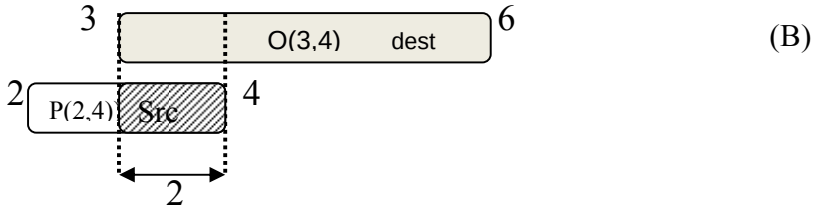


圖 3.12 RLE 模組 B 情況圖

(C)dest完全包含Src情況：當P為(4,1)則與O交集有1長度如下圖3.13所示，故OP的交集的部分從位置4開始到4，則以OP(4,1)表示之。

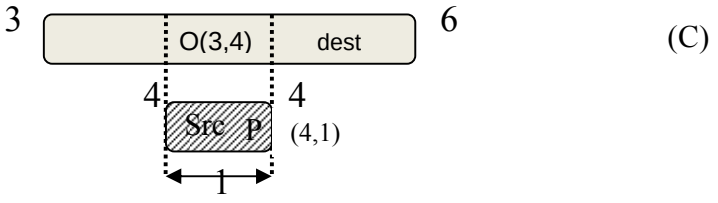
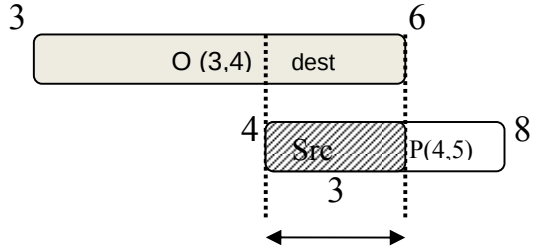


圖3.13 RLE模組c情況圖

第二類：dest_right<Src_right

(D) 部分包含情況：當P為(4,5)則與O交集有2長度如下圖3.14所示，故O的交集的部分從位置4開始到6，則以OP(4,3)表示之。



3.14 RLE 模組 D 情況圖

(F)Src完全包含dest情況：當P為(2,7)則與O交集有4長度如下圖3.15所示，故OP的交集的部分從位置3開始到6，則以OP(3,4)表示之。

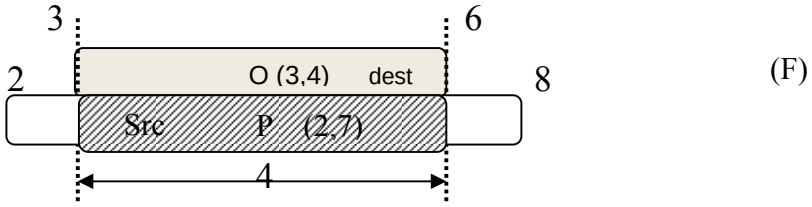


圖 3.15 RLE 模組 F 情況圖

(E)沒有包含情況：當P為(8,2)則與dest比對都無交集的情況如下圖3.16。

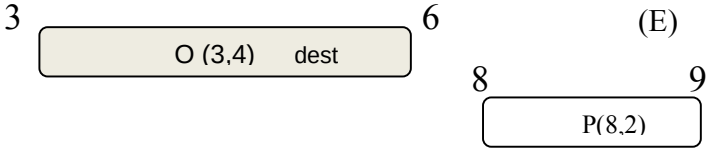


圖 3.16 RLE 模組 E 情況圖

```
Int getSupport(K-itemset S,min_sup)
{
   resRLE←RLE(S(1))
   for(I=2 to k)
   {
      (Count,resRLE) ← And_OP(resRLE,RLE(S(I)))
        If(Count<min_sup)    return -1; // non frequent
   }

    Return count;
}
Public void And_OP(destRLE d,SrcRLE S)
{
   Ctr=0;Sindex=1;Dindex=1;
   DestRight=d[Dindex].base + d[Dindex].length-1;
   DestLeft=d[Dindex].base;
   SrcRight=s[Sindex].base + d[Sindex].length-1;
   SrcLeft=s[Sindex].base;

   While(Sindex<=S.Size    and Dindex<=dsize)
   {
      If(DestRight < DestLeft){nextDRun();Continue;}
      If(SrctRight < SrcLeft){nextSRun();Continue;}

      If(SrcRight <= DestRight)
      {
          If(SrcRight < DestRight)                    //case A
           {
              nextSRun();
           }
          Else if(SrcLeft < DestLeft)                 //case B
           {
               Ctr+=(srcRight-DestLeft+1);
               AndRun(destLeft,SrcRight-DestLeft+1);
               nextSRun();
               DestLeft=SrcRight+1;
               Continue;
           }
          Else                                        //case C
           {
               Ctr+=(SrcRight-SrcLeft+1);
               AndRun(SrcLeft,SrcRight-SrcLeft+1);
               nextSRun();
               DestLeft=SrcRight+1;
               Continue;
           }
```

```
 Else
 {
        If(SrcLeft < DestLeft)                    //case D
          {
             Ctr+=(DestRight-SrcLeft+1);
             AndRun(SrcLeft,DestRight-SrcRight+1);
             SrcLeft=DestRihgt+1;
             nextDRun();
             Continue;
          }
        Else if(SrcLeft <= DestRight)         //casse E
          {
             Ctr+=(DestRight-SrcLeft+1);
             AndRun(SrcLeft,DestRight-SrcLeft+1);
             nextDRun();
             SrcLeft=DestRight+1;
             Continue;
          }
        Else                                        //case F
          {
             nextDRun();
             Continue;
          }
      }
   Return Ctr;
}

Public void nextSRun()
{
   Sindex++;
   SrcRight=s[Sindex].base+s[Sindex].length-1;
   SrcLeft=s[Sindex].base;
}

Public void nextDRun()
{
    Sindex++;
    DestRight=d[Dindex].base +d[Dindex].length-1;
    DestLeft=d[Dindex].base;
}
```

圖3.17 RLE演算法

第四章 以 RLE 模組完整範例說明

本章節將以一個完整範例說明，整合第三章本文所提出 RLE 模組挖掘高頻項目集，以下為整個 RLE 挖堀執行的過程簡述。

步驟一、在 Data Stream 針對所設定的 W(框架)進行前置作業編碼並產生 L_1-itemset 即為 RLE。

步驟二、經由 L_1 產生 C_1 的候選項目集。

步驟三、將 C_2 之 RLE List Set Input RLE 模組 And_op()中執行，產生 L_2 的高頻之 RLE List Set。

步驟四、重複步驟二~步驟三，直到無法產生候選項為止。

4.1 首次探勘框架資料

接下來我們來看一個範例，這個範例主要是說明如何利用兩個k-頻繁項目集的RLE來產生(K+1)-候選項目集的RLE，並對其上的RLE 做頻繁項目集的探勘。表3.2 是一個交易資料庫，包含了12筆交易，七種不同的項目。藉由前述的演算法編碼，可將該資料流轉換為如表4.1 的 RLE 串列。而表4.2 包含所有產生的頻繁項目解集合與其RLE List Set 內容。

初始設定值：W=3Bucket、1Bucket=3筆資料 、Support=3 ；故首次執行時則處理最新的3個Bucket資料的itemset的串列，以下的itemset即為 1-items 的候選項。

表4.1 交易資料之RLE

item	RLE List Set	SupCount
B	[(1,1),(3,2),(7,3)]	6
C	[(1,1),(8,1)]	2
D	[(1,1),(6,3)]	4
A	[(2,1),(4,3),(9,1)]	5
F	[(2,2),(5,2)]	4
G	[(3,1),(8,1)]	2
E	[(3,5)]	5

經過對每一項目的RLE List之每個RUN編碼中的二維位置值做加總，即可快速得到1-頻繁項目集如上表4.1中的C、G 是沒有超過Support值，所以該兩個 items並不會出現在L_1的頻繁項目集內如下表4.2，當我們取得1-頻繁項目集後，我們可經由Apriori的規則來產生多項的頻繁項目集。

表4.2 1-頻繁項目集

Item	RLE List Set	SupCount
B	[(1,1),(3,2),(7,3)]	6
D	[(1,1),(6,3)]	4
A	[(2,1),(4,3),(9,1)]	5
F	[(2,2),(5,2)]	4
E	[(3,5)]	5

接下來由1-頻繁項目集來產生2-itemset的候選集如下表4.3

表4.3 2-候選項目集

2-itemset候選集
BD
BA
BF
BE
DA
DF
DE
AF
AE
FE

以BD為例取B與D的RLE串列即為B=[(1,1),(3,2),(7,3)]、D=[(1,1),(6,3)]，將B與D的串列帶入RLE的演算法中日And_op()函式運算得到BD=[(1,1),(7,2)], 則表示第1位置開始有1個長度及第7位置開始有2長度是交集的部分，故 BD有交集的部分為3長度。下表4.4為2-候選集串列。

表4.4 2-候選項目集RLE

2-itemset候選集	RLE List Set	SupCount
BD	[(11),(7,2)]	3
BA	[(4,1),(9,1)]	2

BF	[(3,1)]	1
BE	[(3,2),(7,1)]	3
DA	[(6,1)]	1
DF	[(6,1)]	1
DE	[(6,2)]	2
AF	[(2,1),(5,2)]	3
AE	[(4,3)]	3
FE	[(3,1),(5,2)]	3

由2-候選集串列經GetSupport()函式可取得2-頻繁項目集如下表4.5

表 4.5 2-高頻項目集 RLE

2-頻繁項目集	RLE List Set	SupCount
BD	[(11),(7,2)]	3
BE	[(3,2),(7,1)]	3
AF	[(2,1),(5,2)]	3
AE	[(4,3)]	3
FE	[(3,1),(5,2)]	3

我們再由2-頻繁項目集產生3-itemset的候選集如表4.6

表4.6 3-候選項目集RLE

3-itemset候選集	RLE List Set	SupCount
BDE	[(7,1)]	1
AFE	[(5,2)]	2

在3-itemset候選集因Support都沒有超過門檻值，故再此W(框架)的資料最多只能探勘到2-頻繁項目集(BD、BE、AF、AE、FE)。

4.2 進行框架位移下一個 Bucket探勘

這部分框架位移使框架內的資料異動以達到模擬資料串流的模式,因此我們在一開始就設定每1Bucket資料量為3，故框架內所含括的資料ID為4~12。

我們必須針對從框架移出的資料來對RLE List Set的item串列做刪除並且對記憶體中的索引值重新配置；再此移出的範圍為第一個bucke資料如下表4.7，涉及的item有B、C、D、A、F、G、E。經對串列移除更新調整後基底項目串列如下表4.8

表4.7 移出為第一個bucke資料

	交易ID	交易資料
第1個bucket	T01	BCD
	T02	AF
	T03	BEFG

表4.8 移出為第一個bucke資料後的更新編碼

item	RLE List Set
B	[(1,1),(4,3)]
C	[(5,1)]
D	[(3,3)]
A	[(1,3),(6,1)]

F	[(2,2)]
G	[(5,1)]
E	[(1,4)]

以B舉例：原B項目的RLE串列為[(1,1),(3,2),(7,3)]此時框架(W)位移量=3，所以即判斷存在陣列中的位置是小於等於3的部分都是必須移除的，如圖4.1所示

（原B項目存在記憶體中的位置）

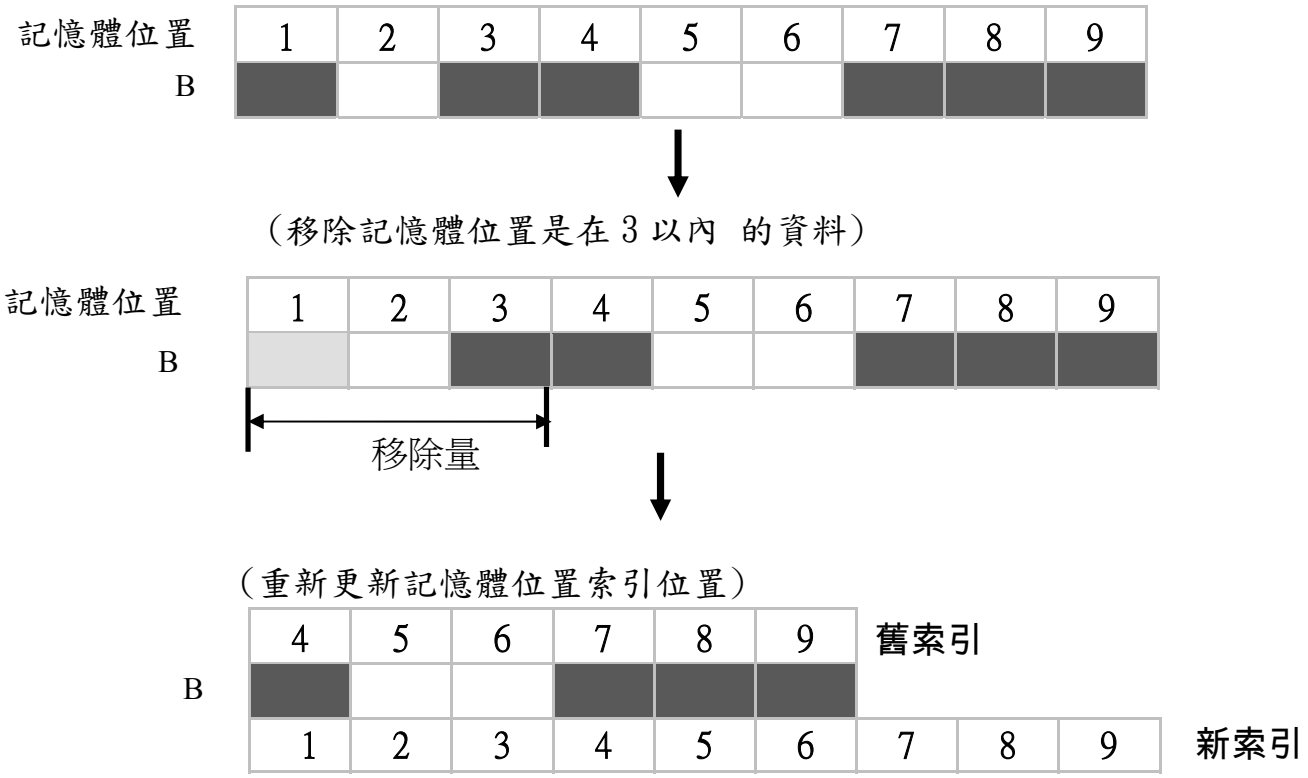


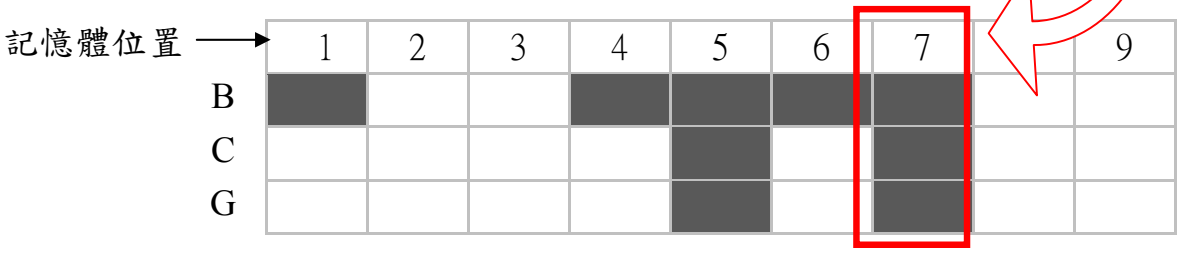
圖 4.1 記憶體中索引值變動情形

接下來針對框架位移後的最新的 Bucket 的資料如表 4.9 做處理，根據上一步驟的串列做新增，如當 T10 寫入時，則存入記憶體位置為 7, 也就是說在記憶體位置 7 會出現 B、C、G 如下圖 4.2 所示

表 4.9 框架位移下一個 bucke 資料

	交易ID	交易資料
第 4 個 bucket	T10	BCG
	T11	AEG
	T12	BG

圖4.2 資料在記憶體中新增情形



故將 T10、T11、T12 寫入產生新的串列項目 1-itemset 候選集如下表 4.10 由新的 1-itemset 候選集得知 1-頻繁項目集為 B、D、A、G、E 並把此項目集當做產生(K+1)頻繁項目集的基底串列，再以 Apriori 演算法精神遞迴的架構一直連續的往最新的資料探勘。

表 4.10 1-itemset 候選集

item	RLE List Set	Support
B	[(1,1),(4,3),(7,1),(9,1)]	6
C	[(5,1),(7,1)]	2
D	[(3,3)]	3
A	[(1,3),(6,1),(8,1)]	5
F	[(2,2)]	2
G	[(5,1),(7,3)]	4
E	[(1,4),(8,1)]	5

由以上的實例說明，我們對探勘資料經前置作業的規則編碼後寫入記憶體並直接對記憶體中的 RLE 作運算比對來產生高頻項目集，隨著框架移動產生有部分資料被刪及新資料流入，在這樣的環境下只須對記憶體中的 RLE List-Set 重新更新並重新探勘。

第五章 實驗結果

在資料流環境中進行探勘，兩個最主要的關鍵因素是執行時間和記憶體的使用量。本實驗主要的重心是放在兩個演算法執行效率的比較上。為了比較本研究所提出RLE 演算和FP-Growth演算法的效能，我們設計了五個實驗項目。

5.1 實驗環境建置

5.1.1 軟硬體環境

硬體：

CPU 2.0G

Memory 2G

軟體：

Microsoft Visual Studio C# 2008

Windows XP Pro

5.1.2 測試資料庫

IBM 產生器 300000 筆資料[16]

本研究採用 IBM Almaden Research Center 所產生的資料，此類資料經常被用來作測試資料庫使用，除了較有公信力外，測試的結果也較能夠與他人研究比較。該產生器在產生資料時有數個參數可供設定，產生測試資料庫的參數分述如下表 5.1。

表 5.1 資料庫參數設定

Ntrans	Tlen	Patlen	nitems	註記
|D|	|T|	|I|	N	Ta.1b.Nc.Dd
資料庫所包含的交易筆數	交易所包含的平均項目個數	資料庫之平均高頻項目集之項目個	資料庫包含總項目個數	如：T10I4N1KD100K 表示此資料庫包含 10 萬筆交易,其中每筆交易的平均項目長度為 10，高頻項目平均長度 4,項目個數為 1000 個。

5.2 交易規則長度編碼(RLE)與 Fp-tree 比較

下圖5.1為RLE與FP演算法的比較。該數據框架大小為10000筆資料，每次位移量為1000筆資料,支持度為1%。圖5.1中的的橫軸為位移的框架數目，即每單位為1000 筆資料。縱軸為總運算時間，將不斷計算每次框架位移導致資料模型改變而花費的探勘時間增加幅度。由圖5.1 可清楚看出RLE花費時間與Fp-Growth 的時間增加幅度明顯小於Fp-Growth，但是隨著資料的流入流出，在每次異動後RLE 所花費的時間相差幅度不太。

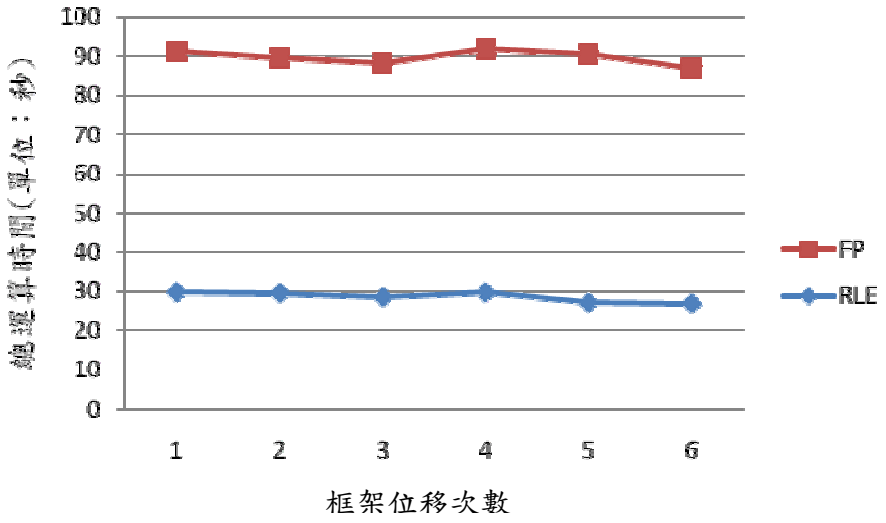


圖 5.1 為 RLE 與 FP 演算法的比較(位移量為 1000)

在下圖5.2 中，我們使用如同圖5.1 的參數，框架大小為100000，支持度為1%，但是每次位移的資料量有所差別。圖5.2採位移量為10000。實驗證明在不同的位移量下，RLE 演算法都將優於Fp-Growth。

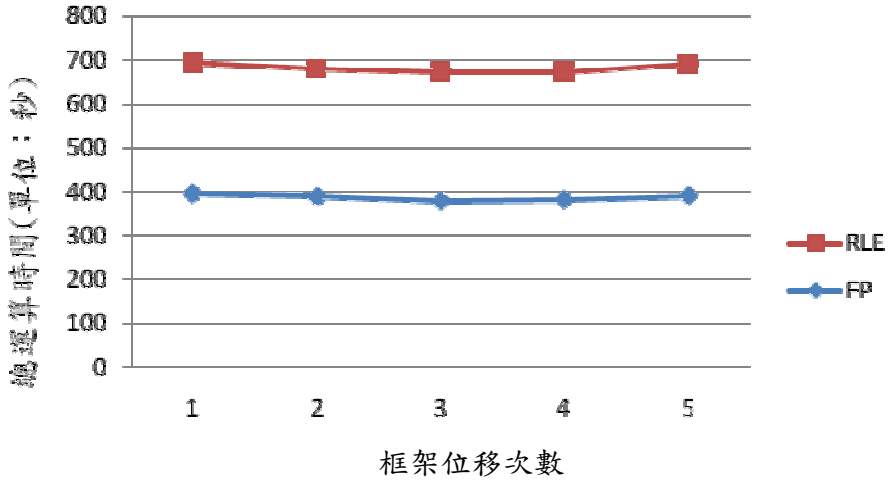


圖 5.2 為 RLE 與 FP 演算法的比較(位移量為 10000)

5.3 RLE 編碼效能評估

以下我們將對RLE 在不同的平均交易長度上，不同的項目數量及支持度作效率評估。

5.3.1 不同平均交易長度下的評估

下圖5.3為在不同的交易長度下所進行的實驗，我們分別在平均交易長度為3、5、10 來進行。實驗中框架大小為100000筆交易，X軸為位移的框架數目，每次位移的資料量為10000筆交易/bucket)，Y 軸為總運算時間，我們累記每次框架位移所探勘花費的運算時間。由圖5.3 得知在平均長度愈長所花費的時間就相對的。

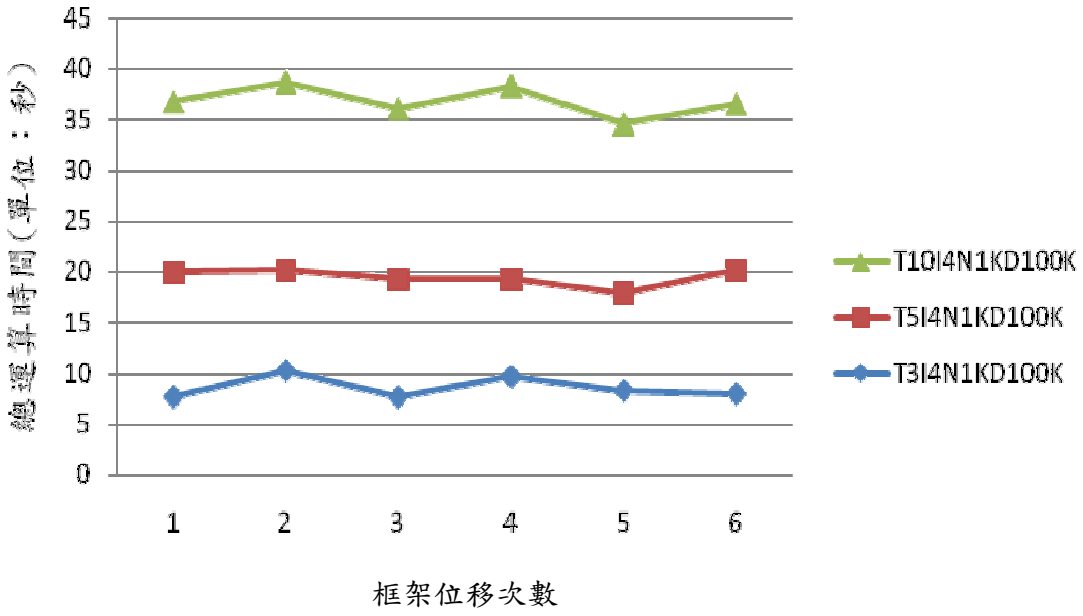


圖 5.3 不同平均交易長度下效率評估

5.3.2 不同項目數量下的評估

我們對不同的項目數量來作評估。圖5.4 為當我們分別在項目數量為1000、1500、2000 進行探勘動作比較的模擬圖例。其中我們預設框架大小為100000，每次位移10000 筆資料當作bucket 大小，支持度為1%。X 軸為每次位移的bucket 個數，Y 軸為每次位移後進行探勘所花費的時間加總。由圖形中我們可以發現在項目數量較大時我們所花費的時間也是相對較多的。由於每個項目都將會透過演算法，產生一串對應的RLE 來進行探勘作動作，項目越多則RLE 會越多。且在每次尋訪所有的RLE 時，我們也需要花相對較多的時間來完成，故這個結果是合理的。

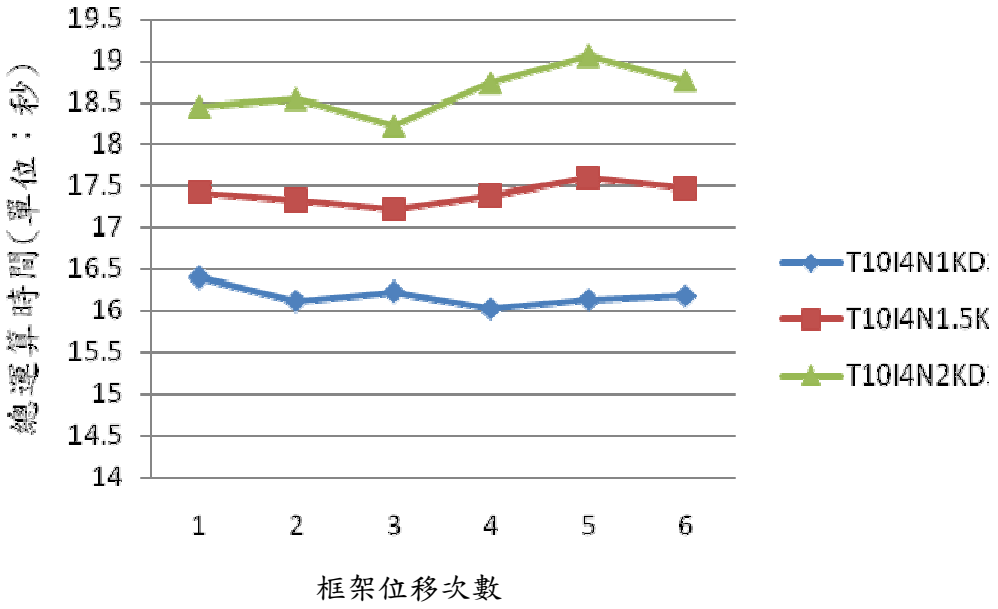


圖 5.4 不同項目下的評估

5.3.3 不同支持度下的評估

最後我們檢視在不同支持度下RLE 的效能評估。圖5.5 是利用資料庫T10I4N100D100k 來進行，該交易資料庫平均交易長度為10，平均高頻項目長度為4，項目數量為100，總共有10000 筆交易。圖中X 軸部份為實驗的支持度數據，Y 軸為運算時間(單位：秒)。我們發現當支持度不斷降低的情況下將會使探勘時間增加。

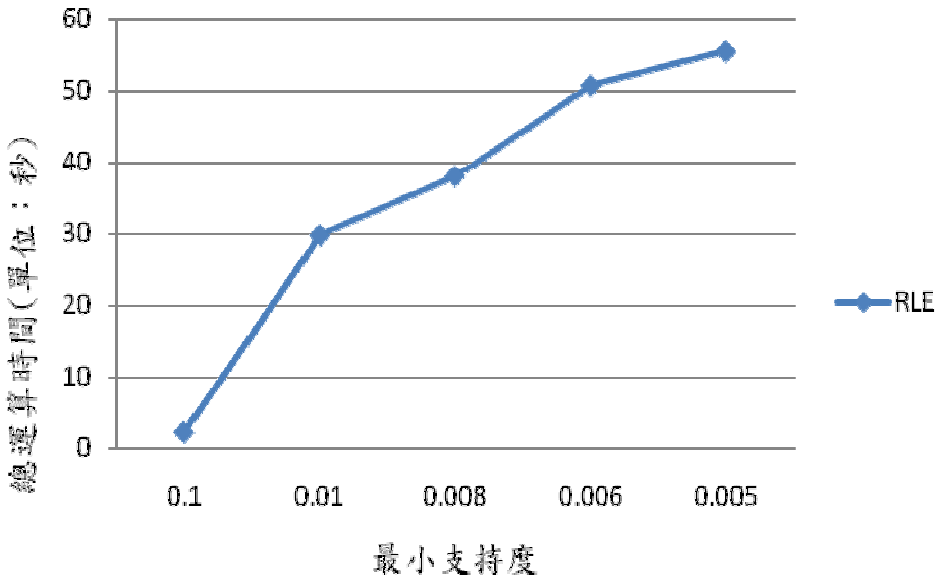


圖 5.5 不同支持度下的評估

5.3.4 不同項目數量下的編碼壓縮量

最後我們檢視在不同項目數量下的編碼壓縮量。圖5.6 是當我們分別在項目數量為1000、2000 進行資料編碼。其中我們預設框架大小為10000還有100000兩種情形。X 軸為資料，Y 軸為編碼壓縮後所佔記憶體空間。由

圖形中我們可以發現在項目數量愈大時經編碼後所佔記憶體空間可以更清楚的顯示出RLE所佔記憶體比Vector更小，因此在整個探勘的過程中不僅不用重新解壓縮編碼並且減少記憶體的使用，因此對於提升整個探勘效率上是個重要的關鍵。

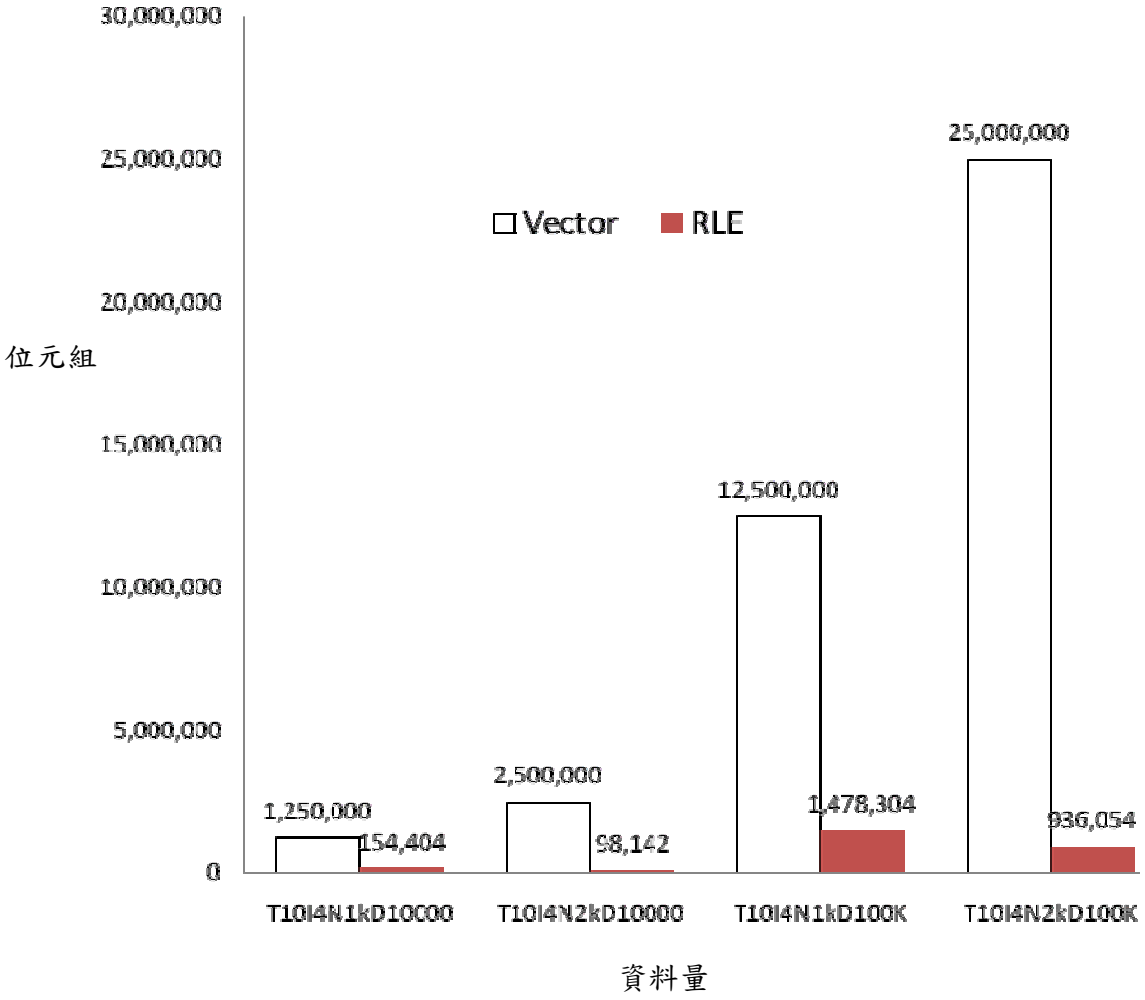


圖 5.6 不同項目數量下的編碼壓縮量


Form1
RunTime :
8.03125
分析完成
D:\論文撰寫資料\程式\第二版整數\Apriori\simon_10000.txt
....
Input Data
Window
3
B
資料總筆數
10000
Bucket
3333
資料
資料總Bucket
3
Support
15
Run Api
4499=15
47063=17
64853=15
72113=16
32435=23
52204=17
55145=28
86834=16
99081=21
11305=17
3678345973
=17
319214000=
15
319219009=
16
319223130=
16
319261424=
16
319214000190
09=15
319214000231
30=15
319214000614
24=15
319214000190
0923130=15
319214000190
092313061424
Move
Next B
Clear
目前處理
1 ~ 3
Bucket
4499=11168164311256113631258813229137711437915852169011693017341174051753415
47063=11644112571136413230135321384614878150701585516414169011740617535178521811119461164853=111051644112571136411396121311323015
Auto Rum
Save
DataBase

圖 5.7 RLE 系統畫面

第六章 結論與未來發展方向

要達到線上挖掘，最直覺的方法就是事先計算出所有項目集的出現次數，但由於記憶體的限制，要儲存所有的項目集之出現次數是不可行的。而為了要在有限的記憶體空間中作線上挖掘，便必須要刪除較不可能會利用到的項目集，而只儲存較可能會利用到的項目集。由於使用者設定門檻值後，系統產生規則時只會參考到高於門檻值的項目集，因此利用預設門檻值來判定項目集的重要性是一個好的方式。

本研究的貢獻，使系統能在有限的記憶體空間中，提供對動態資料庫作線上挖掘關聯式法則的功能。

在本篇論文中，我們提出一個編碼探勘方法，以Apriori理論為基礎，利用啟始位置及長度的數據的方式產生規則的1-items編碼當作基底，再從基底中找出高頻的項目集。

在RLE探勘的方法上，利用相鄰的串列來記錄item是否為連續性的出現，如此以簡單的交集運算便可能得知所有的序列型樣及其對應在記憶體中的item位置，再利用滑動式視窗來模擬資料串流樣式並重新對框架位移量重新建立陣例索引(框架索引)。此方法僅需對資料庫掃瞄一次並建立規則編碼(RLE)；即可從編碼中得知常見1-items，再利用1-items與編碼，快速地找出符合條件的多項高頻繁項目集。

所提出的方法主要以效率為指標，以便減少記憶體的使用量並兼顧減少磁碟讀取的優點並結合規則編碼較有彈性，對於每次異動的資料庫作探勘時

必須重新掃描資料庫,相較之下我們所提出的方法對於異動頻繁資料庫作探勘在效率上有著明顯改進。 由於我們是使用類似Apriori 演算法，當支持度設定很低的時候，我們將會產生大量的1-頻繁項目集，此時利用apriori_gen 來產生的2-候選項目集將會相當龐大,會有大量的運算時間在檢測是否該候選項目為頻繁,故如何縮短檢測2-候選項目的時間將可能是未來的一個挑戰。因此，我們可以針對這點來改進，考慮預先過濾不可能是大型的候選項目集之方法。

參考文獻

[1] S. Thomas, S. Bodagala, K. Alsabti, and S. Ranka, “An efficient algorithm for the incremental updation of association rules in large databases,” Proc. of the 3rd International Conference on Knowledge Discovery and Data Mining (KDD97), 1997, p.p 263-266.

[2] D. W. Cheung, S.D. Lee, and B. Kao, “A general incremental technique for maintaining discovered association rules,” Proc. of the Fifth International Conference On Database Systems For Advanced Applications, 1997, p.p 185-194.

[3] D. W. Cheung, J. Han, V. T. Ng, and C. Y. Wong, “Maintenance of discovered association rules in large databases: An incremental updating technique,” Proc. of the IEEE ICDE'96, 1996, p.p 106-114.

[4] A. Amir, R. Feldman, and R. Kashi, “A new and versatile method for association generation,” Principles of Data Mining and Knowledge Discovery, First European Symposium, PKDD '97, 1997, P.P 221-231.

[5] C. C. Aggarwal and P. S. Yu, “Online generation of association rules,” Proc. of the IEEE ICDE'98, 1998, p.p 402-411.

[6] C.H. Lin, D.Y. Chiu, Y.H. Wu and A.L.P. Chen, Mining frequent itemsets from data streams with a time-sensitive sliding window. In Proceedings of the Fifth SIAM International on Data Mining, Newport Beach,USA,2005.

[7] Richard M.Karp,Christos H.Papadimitriou,and Scott Shanker.A Simple

Algorithm for Finding Frequent Elements in Streams and Bags.Available from http://www.cs.berkeley.edu/ Christos/iceberg.ps,2002.

[8] R. Agrawal, T. Imielinski, & A. Swami, “Mining Association Rules between Sets of Items in Large Database,” Proceedings of SIGMOD, Washington, USA, p.p 207-216 (1993).

[9] R. Agrawal, T. Imielinski and A. Swami, “Database Mining: A Performance Perspective,” IEEE Transactions on Knowledge and Data Engineering, p.p 914-925 (1993).

[10] R. Agrawal and R.Srikant, “Fast Algorithm for Mining Association Rules in Large Databases,” Proceedings of The 20th International Conference on Very Large DataBases, Santiago, Chile, p.p 487-499 (1994).

[11] Han, J., Pei, J., and Yin, Y., "Mining frequent patterns without candidate generation: A frequent-pattern tree approach", Data Mining and Knowledge Discovery, 2004.

[12] 資料探勘.關聯法則p110~P112丁一賢、陳牧言 合著

[13] J.H. Chang and W.S. Lee, Finding Recent Frequent Itemsets Adaptively overOnline Data Streams, Proceedings of ACM SIGKDD, (2003) p.p 487-492.

[14] Yun Chi,Haixum Wang,Philip S. Yu,and Richard R, Muntz. Moment：Maintaining closed frequent itemsets over a stream sliding window. In ICDM, p.p 59-66,2004.

[15] Yun Chi, Yirong Yang,and Richard R. Muntz. Hybridtreeminer：An efficient algorithm for mining frequent rooted trees and free trees using canonical forms.In The 16 th International Conference on Scientific and Statistical Database Management(SSDBM'04),2004.

[16] "IBM Almaden-Quest Data Mining Synthetic Data Generation Code," http://www.almaden.ibm.com/cs/quest/syndata.html

[17] 宋子康，「資料串流中尋找關聯法則之研究」，台灣科技大學訊管理學系，95年碩士論文。

[18] 徐雅琪(2003)，"適用於連續探勘的關聯規則演算法"， 國立台灣科技大學資訊管理研究所碩士論文。

[19] S.J. Yen and Arbee L.P. Chen, "An Efficient Approach to Discovering Knowledgefrom Large Databases", In Proc. of 4th Int. Conf. on Parallel and Distributed- 71 -Information Systems, pp. 8-18, Miami Beach, Florida, USA, December 1996.