南　華　大　學

資訊管理學系

碩士論文

基於連續長度編碼之關聯法則挖掘法

An association rules miner based on run length encoding

研　究　生：謝宗豪

指 導 教 授：邱宏彬　博士

中華民國　97　年　6　月

南　華　大　學

資訊管理學系

碩 士 學 位 論 文

基於連續長度編碼之關聯法則挖掘法

研究生：(簽名)

經考試合格特此證明

口試委員：(簽名)

(簽名)

(簽名)

指導教授：(簽名)

系主任(所長)：(簽名)

口試日期：中華民國 91 年 6 月 18 日

誌　謝

這篇論文的完成，宗豪我真的要感謝指導教授「邱宏彬博士」在這二年以來辛苦的教學、指導；每當小組在研究室開會、討論時，老師都能給予許多建議與分享一些相關經驗讓我不僅可以學得本研究領域等知識且亦學會了其他相關的知識，讓宗豪我能夠得到許多我想了解的技術、與觀念，進而了解如何做研究並且順利的訂定自已所理想的研究方向。

另外也要感謝本小組團隊「204 中隊」的每一個成員的分享與指導，讓宗豪可以在這二年內習得許多先前所不知的觀念與知識，也謝謝在這相處期間都會相互鼓勵、及打氣，給予正向的思考、積極的態度來面對每一個困難與問題。

最後要感謝家人的支持與加油，無論在我個人二年內心情的高低起伏，他們是最清楚不過，但也都一步步的陪我一起走過來，給我很大的空間來發揮，與精神上的支持、鼓勵！更要一提的是，謝謝父親在這二年來對宗豪進修費用支援協助，才讓我有足夠的錢可以念完這學位，宗豪真的很感謝、更會永遠記這份親情，再次謝謝所有的人，謝謝您們！

基於連續長度編碼之關聯法則挖掘法

學生：謝宗豪　　指導教授：邱宏彬博士

南　華　大　學　資訊管理學系碩士班

摘　　要

說到關聯規則是這幾年來非常熱門所討論的話題，在各行各業中也都慢慢的重視關聯規則的重要性與實用性，然而在演算法的處理執行效率將是大家所關心的重點。在早期在關聯規則演算法是將會員資料庫相關交易記錄存在資料庫中，然後再進行資料探勘，因為資料處理都在硬碟中運作，所以在效率處理上，執行速度都不是很理想，相對的降低了資料探勘的效率與速度。

我們所提出的方法即可以改變上述的面臨的問題，方法為「連續長度編碼方法(Run-length Encoding RLE)」，用編碼技巧方法進在主記憶體中壓縮資料，進行探勘，在其探勘過程無須解壓縮編碼，進而增加探勘之效率。

關鍵字：資料探勘、關聯規則、資料壓縮

An association rules miner based on run length encoding

Student：Hsieh Tsung Hao Advisors：Dr. Hung-Pin Chiu

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

ABSTRACT

It is very hot topics discussed over these several years to talk about the related rule, importance and practicability with related and regular attention slowly too in all trades and professions, but it will be a focal point that everybody cares about that the treatment performing the algorithm carries out efficiency.

It stores member's database relevant trade record in the database to perform algorithms in the related rule in early days, then carry on materials prospect, because materials punish operation of the hard disk, so in efficiency is dealt with, it is not all very ideal to carry out the speed, efficiency and speed that the relative reducing materials prospect.

Question the method can change above-mentioned facing put forward by us, the method is ' coding method of the continuous length (Run-length Encoding RLE) ',it is by code skill enter method because mainly whether among storing device compress by materials, it is prospect not to go on, needn't be decompressed and encoded to prospect course in it, and then increase the efficiency prospected!

Key words：Data Mining, Association Rules, Data Compression

目　　錄

表　　目　　錄

圖　　目　　錄

第一章　緒　　論

1.1　研究背景

在這個資訊科技不斷更新的時代，企業所面臨的是與日俱增的競爭壓力與資訊爆炸所帶來的衝擊，雖然每天資料量不斷的再增加，但是企業如果不能從這些龐大且複雜的資料中獲得有用的資訊，就無法讓公司更加具有競爭力，所以無限的商機就存在企業內、外部的資料庫或資料倉儲中，因此「資料探勘」這個技術就此產生，藉由資料探勘的方式，來尋找對企業有幫助的資訊，並利用其作為對決策的參考工具。

在這些豐富的資料中事實上隱藏許多有用的資訊，例如大賣場或百貨公司可以藉由交易資料庫找到消費者購買習慣，做為行銷策略的依據等；為了從資料庫中找出這些有用的資訊，由於資料量的龐大，因此無法利用人工具接進行分析與處理，故必須運用一些適當的方法將其找出來，於是資料探勘(Data Mining)技術將被許多人所重視。

1.2 資料探勘的分類

資料探勘的技術，可依資料庫的型態與應用的領域來加以分門別類，一般可分為：關聯規則(Association Rules)、分類(Classification)、群組化(Clustering)以及序列型樣(Sequential Patterns)等，藉由上述的方法，故可以對資料進行各種分析以取得對使用者有意義的資訊。我們將介紹上述的各類方法。

(1) 關聯規則(Association Rule)：

關聯規則的相關技術主要是用來找出資料庫中的項目種類(Item)間的關聯性，以便形成關聯規則。由於關聯規則具有描述項目種類間的關聯特性，因此，它能用以找出並顯示不同商品間的銷售關聯性與客戶消費傾向等資訊，運用這些資訊，使用在大型賣場，可以重新安排商品陳列的方式，將顧客常會一起購買的物品盡可能的放在鄰近的位置，以刺激消費者購買力及增加賣場營業獲利。

(2) 分類(Classification)：

分類技術是指預先建構一套分類的準則，將資料依照這些準則進行分類。分類使用監督式的學習技術，用事先定義好的類別來進行資料的分類[8]。分類時可以先使用資料庫中的部分資料來訓練，訓練後學習到的分類能力就可以進行整個資料庫的分類工作。目前一

般較為普遍使用的分類方法是利用決策樹(Decision Tree)來做分類器。

(3)群組化(Clustering)：

分群技術是不事先建構一套分類的標準，剛是依照資料的特性，將性質相似的資料歸於同一群。分群是指使用非監督式的學習方式，將具有相同趨勢或型樣的原始資料區分成群。分群之後，各集群內部的差異要小，而群間差異要大。使用的技巧包括 K-means 法及 agglomeration 法。

(4)序列型樣(Sequential Patterns)：

挖掘序列型樣是資料探勘的一個研究方向，它可以用在發掘顧客的購買行為上，找出大部分顧客在不同交易中採購物品的先後順序之行為，主要的目的是從顧客的交易資料，找出有用的資訊，這些有用的資訊可以運用在行銷的促銷策略上。根據 Agrawal and Srikant 等人的定義，序列型樣探勘工作就是要找出隨著時間或是特定順序而經常發生的型樣。

在上述這些方法中，關聯規則的探勘是近幾年來最被廣泛研究的議題，關聯規則已被運用於實際的企業應用中，所得到的成效非常的顯著。

1.3 研究目的與動機

關聯規則(Association rule)是資料探勘領域中很重要的一部份，Apriori 演算法是在 1994 年由 R.Agrawal 等人提出，也對關聯規則有著很大的貢獻！不過因為此一演算法有些問題點存在，而有許多學者進而再提出一些改進 Apriori 在找尋高頻項目組(Large Itemsets)時會產生大量的候選項目集、與每產生一次高頻項目集時都要掃瞄一次資料庫等缺失的方法。

在早期在關聯規則演算法是將會員資料庫相關交易記錄存在資料庫中，然後再進行資料探勘，因為資料處理都在硬碟中運作，所以在效率處理上，執行速度都不是很理想，相對的降低了資料探勘的效率與速度。

我們所提出的方法即可以改變上述的面臨的問題，方法為「基於連續長度編碼之關聯法則挖掘法(Run-length Encoding RLE)」，用編碼技巧方法進在主記憶體中壓縮資料，進行探勘，在其探勘過程無須解壓縮編碼，進而不影響探勘效率。

1.4 論文架構

本研究方法的目的為提出基於連續長度編碼之關聯法則挖掘法，主要包含兩個部分：

(1) 編碼前置處理階段：

我們提出一個編碼前置處理機制，將交易資料庫轉成特定的資料結構，我們稱之為「Run-Length Encoding(RLE List Set)」

(2) 資料探勘階段：

為我們所提出的資料探勘演算法，主要說明如何利用編碼內容來進行資料探勘並求得支持度。

結合前置處理階段與資料探勘階段就是我們提出的一套基於連續長度編碼之關聯法則挖掘法，這個部分我們在第三章會特別的詳細說明。在第四章中將以著名的 Fp-Growth 與我們所提出的方法來作實驗比較。

本論文的結構如下，第二章為文獻探討，內容為描述一些各類關聯規則演算法的特性與優缺點。第三章為本研究，運用在關連規則資料挖掘演算法，此方法以有效率的方式找尋所有高頻項目組。第四章則是本研究的實驗結果與效能評估部分，其主要在描述本方法與其他關聯規則演算法之比較。第五章則是對本研究結果所做的結論。

第二章 文獻回顧與探討

關聯規則(Association Rule)探勘方法首先由 Agrawal 等人,於 1993 年首先提出，關聯規則的探勘主要目的是在交易資料庫中挖掘出隱藏的資訊，並藉以找出所有的關聯規則。此關聯規則意味著從資料庫中透過某種方法找出來的一則潛在資訊，這種資訊提供了顧客消費的趨勢、傾向和喜好。如果能夠充分掌握顧客的行為就可以帶來更多的商機。例如在商店交易資料中發現消費者所購買的商品中，牛奶和麵包這兩種商品經常被一起購買，而這兩種商品會一起被購買的交易占總交易資料的 6%，並且知道在購買麵包的情況下會再買牛奶的比率為 86%，就可以推測當顧客購買麵包時，有 86%的可能性會一起購買牛奶，這種推論就是一種關聯規則。商家會利用這種資訊，將顧客會常一起購買的商品做適當的搭配，以獲得豐厚的利潤。

將一個由字母符號所組成的集合 $I=\{I_1, I_2, I_3 \cdots., I_n\}$，稱之為項目(items)，然後由I中的子集合所組成的集合稱之為交易(Transaction)，之後這些交易的集合就組成一個資料庫D。每一筆的交易，記錄了該次交易中有哪些項目(Item)產品被購買，且每一個交易都有一個唯一的識別編號，稱之為TID。如表 2-1 即為一個交易資料庫的例子。一群項目的組

合稱為項目組(Itemset)，其中項目組含有一個元素的稱為 1-項目組(1-Itemset)、有兩個元素的稱為 2-項目組(2-Itemset)…以此類推。

表 2-1：交易資料庫

TID	購買項目
T10	EFG
T20	ABF
T30	ABD

在關聯規則的擷取上，一般都分成兩個階段討論，第一階段為找出所有的高頻項目組(Frequent itemsets)，當某一個項目之支持度大於一個人為訂定的最小支持度參數時，該項目組稱為高頻項目組(Frequent Itemset or Large Itemset)，意指該項目組經常地被購買。交易資料庫。的所有滿足最小支持度限制的高頻項目組就稱為完整高頻項目組(Complete Large Itemsets)或直接稱為所有高頻項目組。對於支持度的計算公式，如公式一所示；

$$\text{Support(I)} = \frac{\text{項目組 I 在資料庫中出現的次數}}{\text{資料庫的總交易筆數}} \qquad \text{（公式一）}$$

在產生高頻項目組中，相關項目在資料庫中出現的頻率必須要大於或等於某一使用者所定之最小支持度限制，滿足此條件限制的便稱之為高頻項目組。因為若其出現的頻率太低，則其相對的便比較沒有意義，這樣的項目組對使用者來說是不需要的，使用者比較在意的只是出現次數較

多的項目組。

第二階段為產生關聯規則，對於一個高頻項目組Ｉ，我們可以列舉所有 X→Y 的型態，其中Ｘ、Ｙ⊂Ｉ，且Ｙ＝Ｉ－Ｘ，且Ｘ∩Ｙ＝φ 這樣的 X→Y 即為一條可能的關聯規則，提供買了Ｘ則會買Ｙ的訊息。接著驗證規則的信賴度，每一個可能的關聯規則 X→Y 都有一個信賴度值(Confidence)表示買Ｘ則會買Ｙ這樣的資訊的真實程度。對於信賴度計算的公式，如公式二所示：

$$\text{Confidence}(X \rightarrow Y) \quad \frac{\text{項目組 } X \cup Y \text{ 在交易資料庫中出現的次數}}{\text{項目組 } X \text{ 在交易資料庫中出現的次數}} \qquad \text{（公式二）}$$

當 X→Y 的信賴度大於使用者訂定的最小信賴度值時，這條 X→Y 規則便是一條有興趣的關聯規則，提供買了Ｘ則會買Ｙ的資訊。

給定一個交易資料庫Ｄ，兩個參數為最小支持度與最小信賴度。關聯規則演算法就會從交易資料庫中找出所有滿足要的關聯規則。最小支持度參數的目的在於讓找出來的規則更實用，而最小信賴度參數的目的是讓規則具有某種程度上的可信度。

在第一階段中，這個步驟會花費大量的計算和記憶體。因此對於關聯規則演算法，大多數的學者都致力於改進產生高頻項目組的效率。然而第二階花費的計算與記憶體仍然很多，但是相對上方法較為固定簡單。

因此我們發展的方法也將計對第一階段來處理，集中探討如何能夠有效率的找出所有高頻項目集合。

2.1 Apriori 演算法

Apriori 演算法[8]是由 Agrawal 在 1994 年提出，利用簡單且循序漸進的方式，在資料庫中找出關聯規則。藉由讀取資料庫中的所有交易，來找出出現次數頻繁的項目組，稱為高頻項目組(Frequent Itemsets)，再利用這些高頻項目集且產生關聯規則(Association Rules)。

Apriori 是利用反覆讀取整個資料庫的方式，由短的項目組產生長的項目組，直到找出所有的高頻項目組，並利用所找出的高頻項目，尋找關聯規則。圖 2-2 說明 Apriori 演算法的流程。圖中可以看出 Apriori 演算法是由多次回合而組成，每一個回合的兩個主要階段為 1.產生候選項目組。2.掃描資料庫以決定高頻項目組。每一回合 k 都會產生長度為 k 的高頻項目組，稱為高頻 k-項目組，圖 2-1 為 Apriori 演算法。

1. L_1={large 1-itemsets}

2. for (K = 2 ; $L_{K-1} \neq 0$; K++) do begin

3. C_K= apriori-gen (L_{K-1})
4. for all transactions t $\in$ D do begin
5. C_t=subset (C_K, t)
6. for all candidates c $\in$ C_t do
7. c.count ++;
8. end

9. L_K={c $\in$ C_K| c.count $\geq$ minsup}

10. end

11. Answer = $\cup_K L_K$;

圖 2-1：Apriori 演算法

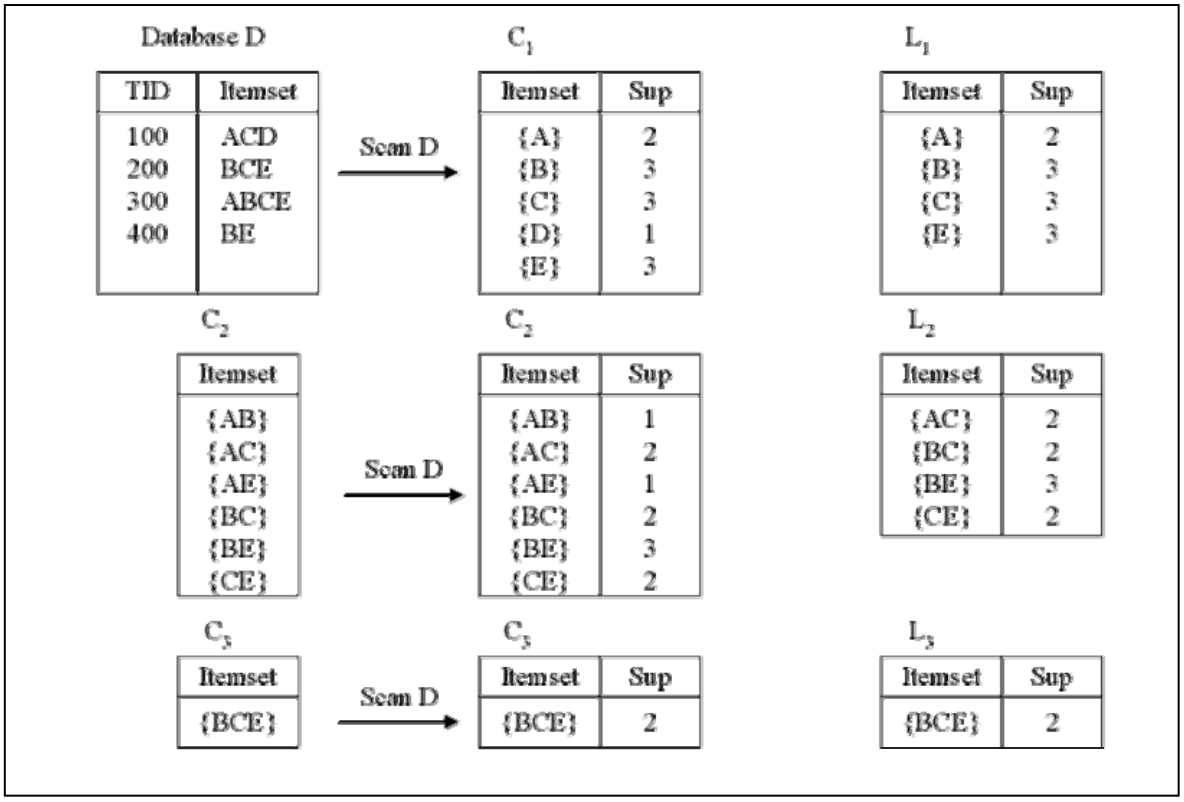

圖 2-2：Apriori 演算法流程圖

Apriori 演算法的兩個瓶頸

(1)產生大量的候選項目集(Itemset)

在產在 2-候選項目時是由 1-頻繁項目兩兩合併產生，若 1-頻繁項目集中有 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-頻繁項目集中有 1000 個項目，則產生 45 萬個 2-候選項目。

(2)需要多次掃瞄資料庫

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瞄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，產生頻繁項目集時所需花費的時間。

綜合以上的問題，利用 Apriori 的資料挖掘方式，以 I/O 成本的觀點來看，是相當不經濟的。

2.2 Frequent-Pattern tree（FP-Growth 演算法）

FP-Growth 演算法是由 Han, Pei, and Yin（2000）提出，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，找出符合最小支持度 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）第二階段探勘 FP-Tree：

（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，其缺點為，所運用到的資料結構較複雜。

2.3 DLG（Direct Large itemsets Generation）演算法

DLG[11]演算法使用 Bit-Map 來計算支持度，以減少磁碟存取時間，另外採用圖形方法來產生候選項目集合。

所謂的 Bit- Map，是將每個項目在資料庫每筆交易中出現與否的情形，利用 0 與 1 的來表示，其中”0”代表沒有出現，”1”代表有出現。所以資料庫中有幾筆交易，則每個項目的位元向量就會有多長，然後將所有項目的位元向量組合起來就成為 Bit-Map。在計算支持度時，只需將要計算的項目之位元向量作相對應位元之”AND”運算，然後計算結果中”1”的出現次數即為支持度。

DLG 在產生長度大於 2（不含 2）的候選項目集合時，所採用的是跟 Apriori 不一樣的方式，DLG 是採用圖形的方式來產生候選項目集合。DLG 在產生 L 後，將所有 L 利用一個有向圖（Directed-Graph）來表示，當有一個{X,Y}為 L 時，在有向圖中就利用一個從項目 X 到項目 Y 的邊來表示這個大項目集合。

DLG 演算法：

i. 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)。

ii. 從第一個步驟中，任意選取兩個項目I_m I_n ($m < n$)，對這兩個項目的位元向量中的每個相對應的位元做 “AND” 運算，然後計算”AND”後結果中有多少個”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

iii. 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。

iv. 產生 K-大項目集合 (K > 2) 時，利用這個有向圖來產生候選項目集合，並使用 Bit-Map 來計算其支持度。在產生C 時K就利用 L 在有向圖中最後的一個項目來做延伸，假設 L =K-1 K-1{i_1 ,i_2,…,i_k }，從 i 有邊

連到 u 這點，此時可產生一個新的1 2 K-1 K-1候選項目集合C ＝ { i_1 ,i_2,…,i_k ,u}，然後將候選項目集合中所K 1 2 K-1有項目的位元向量做 AND運算，可以得到這個候選項目集合的支持度，當此候選項目集合符合條件時，則此候選項目集合就是一個 K-大項目集合（K-Large Itemsets)。

v. 重複第四步驟直到沒有新的候選項目集合產生為止。

以下為DLG範例：

設支持度為 = 2

表 2-2：DLG 交易資料庫

TID	Items（項目）
1	A C D
2	B C E
3	A B C E
4	B E

表 2-3：DLG中L_1和Bit-Map

L_1	位元向量（Bit-Vector）
A	1010
B	0111
C	1110
E	0111

C_2 ={{A B}, {A C}, {A E}, {B C}, {B E}, {C E}}

L_2 ={{A C}, {B C}, {B E}, {C E}}

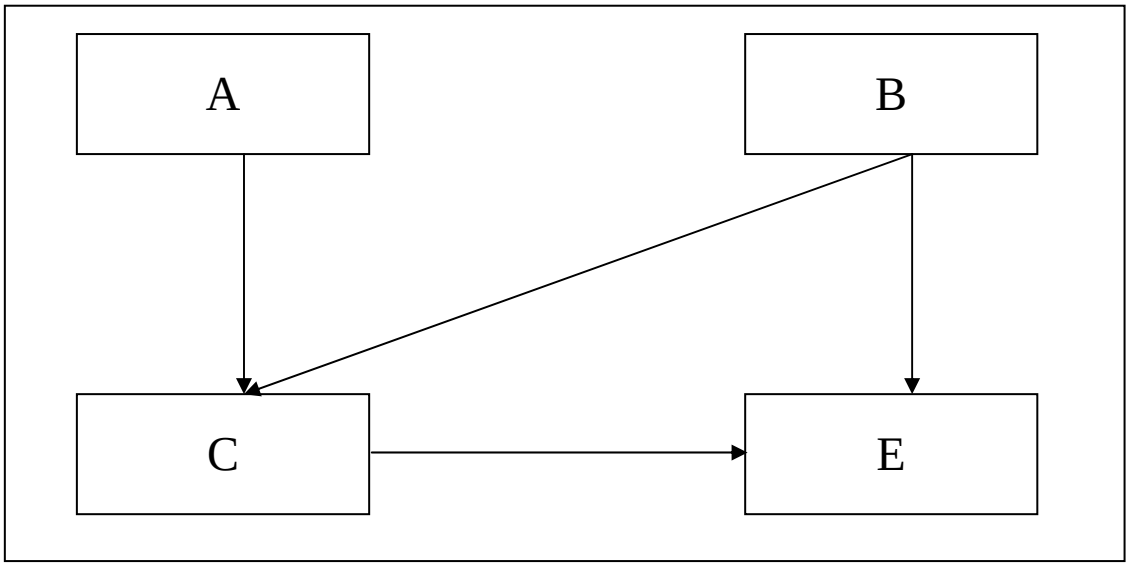

圖 2-3： DLG 關聯圖

C_3 ={{A C E}, {B C E}}

L_3 ={B C E}

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。

以上所介紹之演算法優缺點請參考表 2-4。

表 2-4：各演算法之優缺點比較表

	優點	缺點
Apriori	演算法邏輯簡潔易懂	・多次掃瞄資料庫 ・更改最小支持度時需再重掃瞄 DB ・存於資料庫，硬碟運作，效能不好
FP Growth	只掃瞄資料庫二次、不需要建立候選項	・交易資料量大時，非常佔記憶體空間 ・更改最小支持度時需再重掃瞄 DB
DLG	只需掃瞄資料庫一次產生候選項容易	・交易資料量大時，非常佔記憶體空間

第三章 研究方法

針對上述相關演算法其較弱的地方，本研究提出一套基於連續長度編碼之關聯法則挖掘法，透過前置處理的機制將交易資料庫，轉成以項目出現頻繁次數序列為主的交易連續長度的編碼方式(Run-Length Encoding)，來替代所有的交易紀錄。此份交易紀錄經過轉換過後的編碼資料所需空間將遠小於原本傳統靜態資料庫模式的儲存空間。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，而不需要再對整個原始資料庫進行掃瞄。在計算支持度時，也不需要將編碼作還原原始資料內容的動作，即可求出正確及符合條件的支持度資料。

對於我們所提出的演算法將分成二個部份來進行說明，第一部份為編碼前置處理，主要是將交易資料庫轉成特定的資料結構；第二部份為我們所提出的資料探勘演算法，主要說明如何利用編碼內容來進行資料探勘並求得高頻項目集。

3.1 RLE 資料探勘演算法系統流程說明

我們所提出的基於連續長度編碼之關聯法則挖掘法，稱之為「Run-Length Encoding（RLE)」，此演算法探勘方式與 Apriori 類似。首先會先產生侯選項目集，然後再計算這一些候選項目集的支持度是否符合最小支持度，如果符合將保留下來，否則將其刪除。

關於 RLE 系統流程如圖 3-1：RLE 系統流程圖所示。

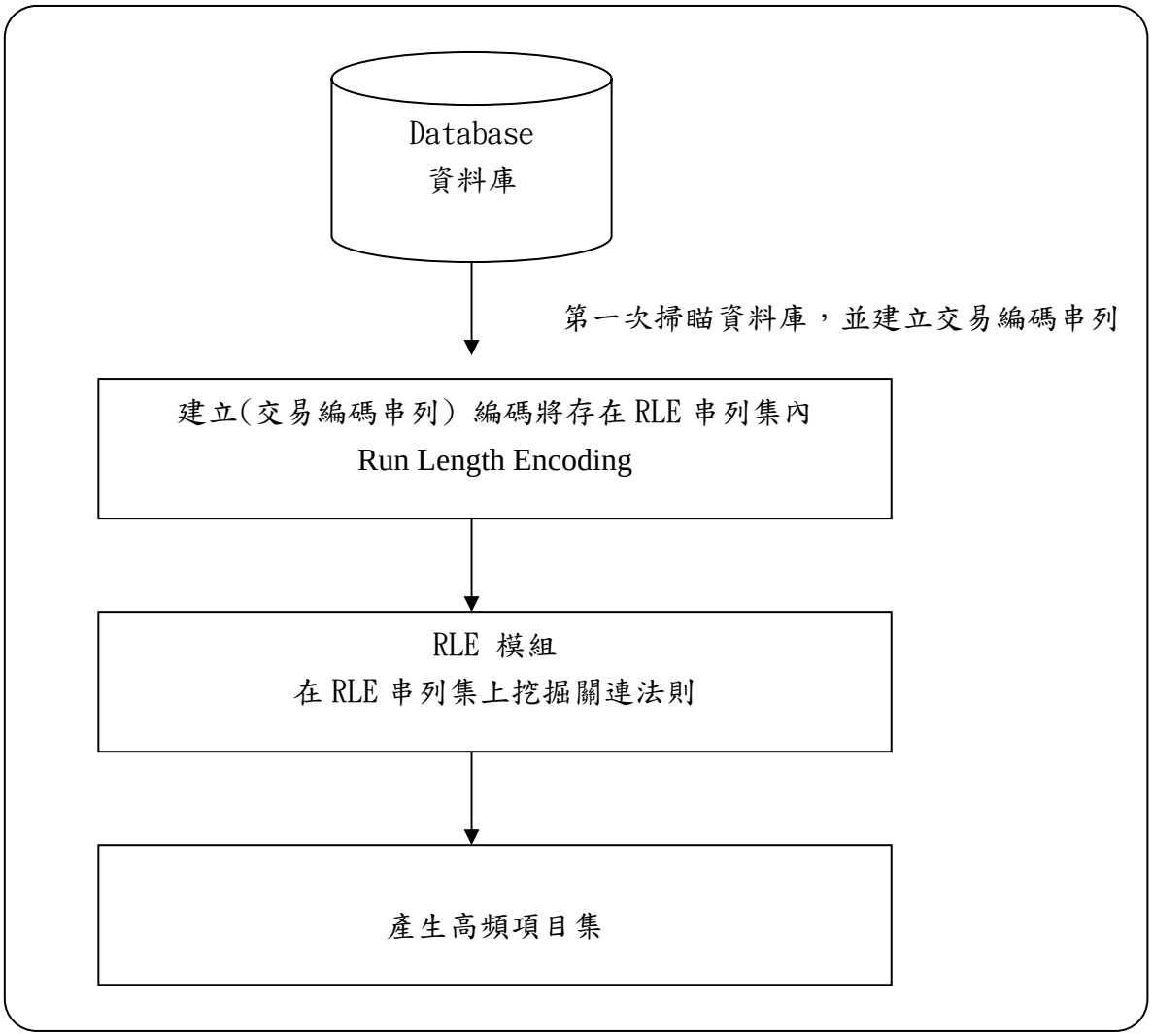

圖 3-1：RLE 系統流程圖

說明 圖 3-1 RLE 系統流程圖：

一、 首先先讀取第一次資料庫，建立編碼規則，並計算所有項目的支持度，再將支持度大於最小支持度(min sup)的L_1高頻項目，再將L_1高頻項目之(交易編碼串列) 編碼將存在RLE串列集內 Run Length Encoding。

二、 利用L_{k-1}產生C_k，然後使用RLE模組方法來計算支持度，將不符合最小支持度的候選項目集刪除，並產生L_k高頻項目集。 重複第二步驟，直到無法產生新的候選項目集為止。

3.2 初始 RLE 串列集的編碼建構

在進行對資料庫交易資料探勘前，必須先建構好交易資料編碼，以下為建構編碼之定義：

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(交易編碼串列)

若項目 I 在交易中共出現 k 個 Run，其 RLE 為：

(t1, ℓ 1),(t2, ℓ 2)．．．．(tk, ℓ k)

則項目 I 的 RLE List 定義為：[(t1, ℓ1),(t2, ℓ2)...(tk, ℓ k)]

如表 3-1、表 3-2 是交易資料範例說明

表 3-1：原始交易資料庫

Tid	Item set
T1	B, C, D
T2	A, F
T3	B, E, F, G
T4	A, B, E
T5	A, E, F
T6	A, D, E, F
T7	B, D, E
T8	B, C, D, G
T9	A, B
T10	A, B, E, F

表 3-2：交易項目之編碼列表

交易項目	每個 Run 的 RLE
A	[(2,1),(4,3),(9,2)]
B	[(1,1),(3,2),(7,4)]
C	[(1,1),(8,1)]
D	[(1,1),(6,3)]
E	[(3,5),(10,1)]
F	[(2,2),(5,2),(10,1)]
G	[(3,1),(8,1)]

如表 3-2，以 B 為例說明，該交易項目的位置及長度為 (1,1),(3,2),(7,4)，如定義一所示：(1,1)表示在第一筆交易有出現 B 的項目有 1 次，則長度為 1；(3,2)表在第三筆交易、與第四筆交易中皆有出現 B 的項目，所以表示從第三筆資料開始連續二筆資料都有出現，長度為 2，(7,4)表示在第七、八、九、十筆交易中皆有出現 B 的項目，所以表示從第七筆資料開始連續四筆資料都有出現該項目，長度為 4。 因此 B 項目的 Run list 為[(1,1),(3,2),(7,4)]，如定義二所示，此 Run List 即是交易編碼串列。

3.3 RLE 演算法模組架構說明

系統模組架構圖如圖 3-2 所示：

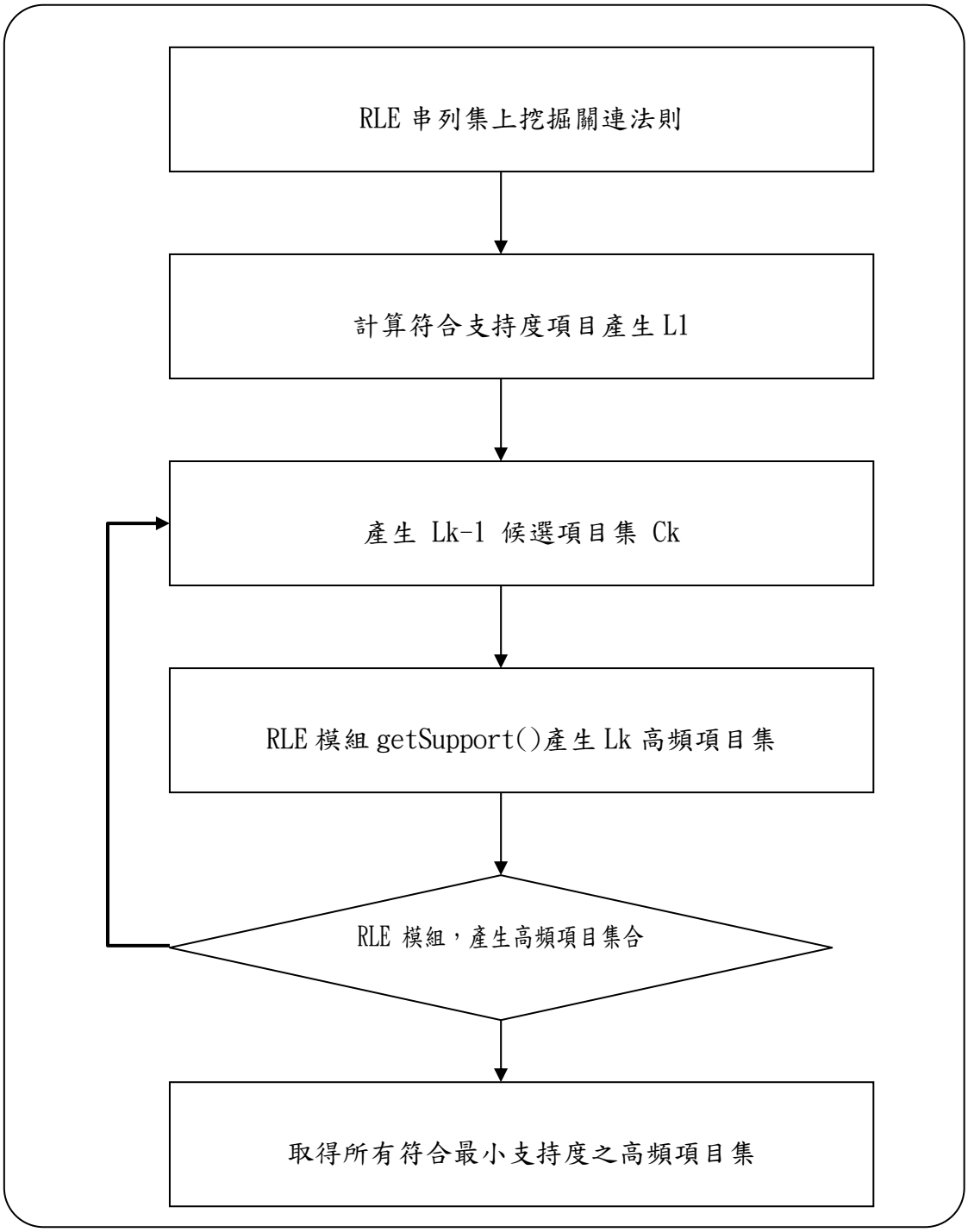

圖 3-2：系統模組架構圖

3.3.1 RLE 進行交集之六種情形表示圖

我們提出一個直接在 RLE 串列集上進行挖掘的演算法，如下圖 3-3，將交易資料編碼後，再以 RLE 模組來判斷二個項目之比對交集之情形，來產生 2 項目之候選集，或是多項之高頻項目集。在圖 3-3 中我們把所有可能項目交集的情形分為二種狀況，狀況一為 A、B、C 之情形，狀況二為 D、E、F，dest 項目與 Src 比對情況可能有下圖之 A、B、C、D、E、F，六種情形，例如：dest、與 Src 沒有交集的情形則為 A、F 情形，其餘皆有部份、或完整交集，以下我們將針對 RLE 模組來說明這六種交集情形，以計算該交集之長度。

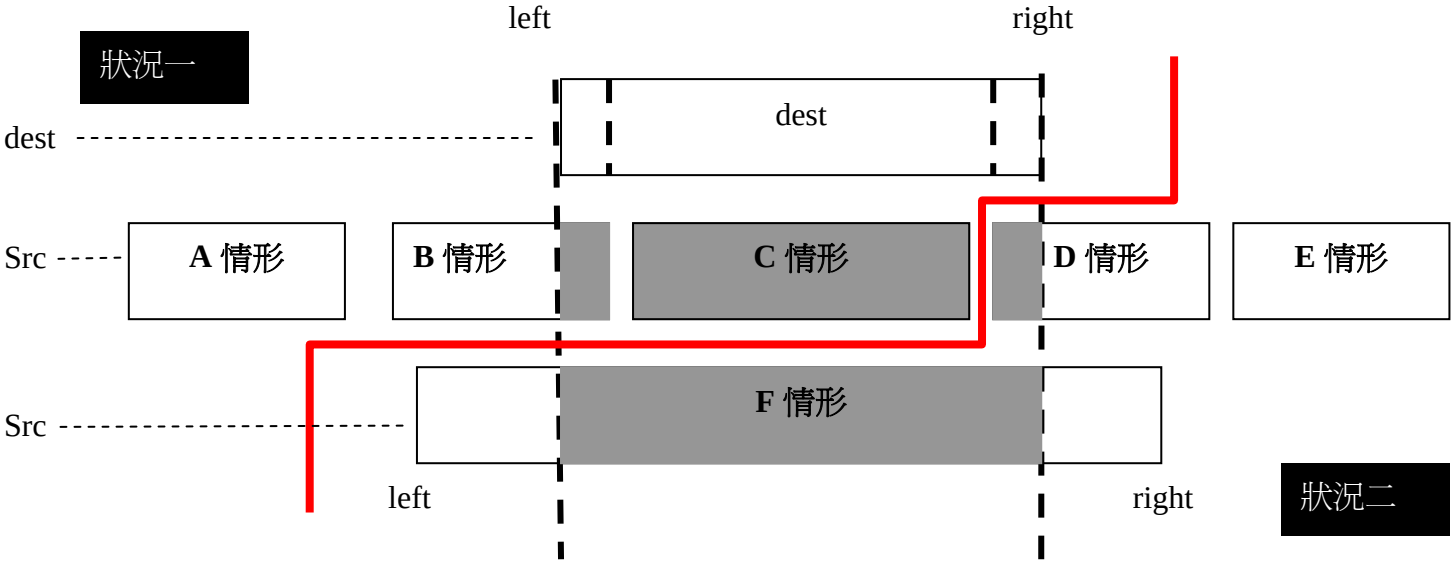

圖 3-3 ：RLE 模組交集情形表示圖

假設項目 A 的 Run1 為(dest)，項目 B 的 Run1 為(Src)，一開始我們則以 dest_right 與 Src_right 進行比較，如 dest_right >= Src_right 則屬於狀況一之 A、B、C 情形，否則 dest_right < Src_right 則屬於狀況二之 D、E、F 情形。

假設：dest 之編碼為(5,3)、Src 之編碼為(1,1)

狀況一的情形為：dest_right >= Src_rihgt

A 情形=沒有交集之情形：當 A 為(1,1)時與 dest 比對則無任何交集之情形，如圖 3-4

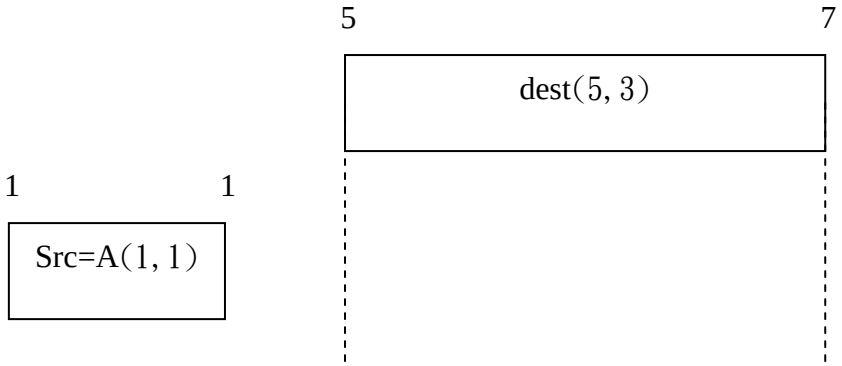

圖 3-4：RLE 模組 A 情形(無交集)

狀況一的 B 情形 = 部份交集情形：dest 之編碼為(5,3)、Src 之編碼為(4,2)，B(4,2)則與 dest 交集長度為 2，所以 dest,B 交集的部份，從位置 5 開始到 6，則以 dest,B(5,2)來表示，如圖 3-5

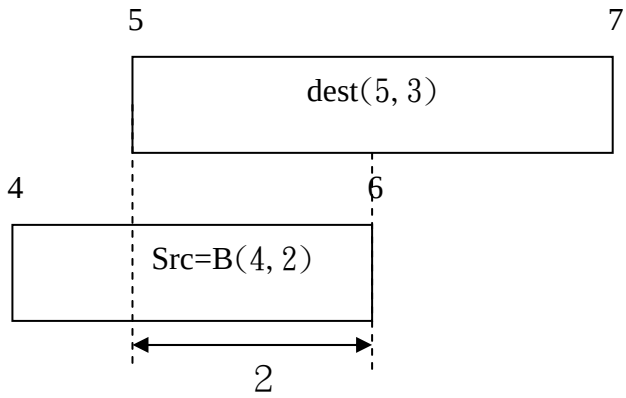

圖 3-5：RLE 模組 B 情形(部份交集)

狀況一的 C 情形 = 完全交集情形：dest 之編碼為(5,3)、Src 之編碼為(6,1)，C(6,1)則與 dest 交集長度為 1，所以 dest,C 交集的部份，從位置 6 開始到 6，則以 dest,C(6,1)來表示，如圖 3-6

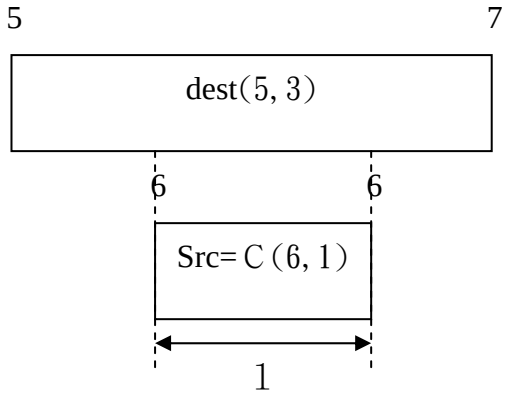

圖 3-6：RLE 模組 C 情形(完全交集)

狀況二的 D 情形 =部份交集情形：dest 之編碼為(5,3)、Src 之編碼為(6,3)，D(6,3)則與 dest 交集長度為 2，所以 dest,D 交集的部份，從位

置 6 開始到 7，則以 dest,D(6,2)來表示，如圖 3-7

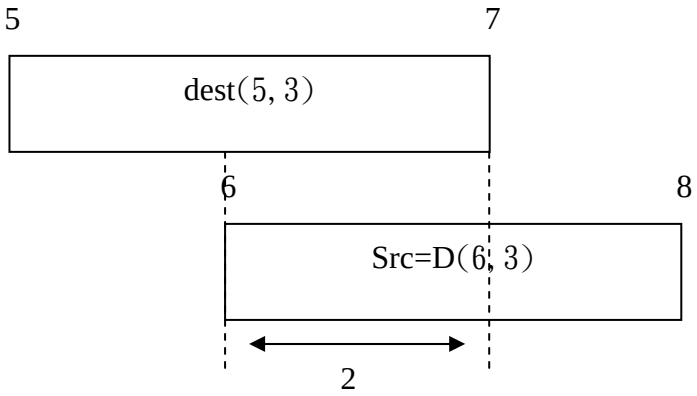

圖 3-7：RLE 模組 D 情形(部份交集)

狀況二的 E 情形 = 沒有交集情形：dest 之編碼為(5,3)、Src 之編碼為(8,2)，如圖 3-8

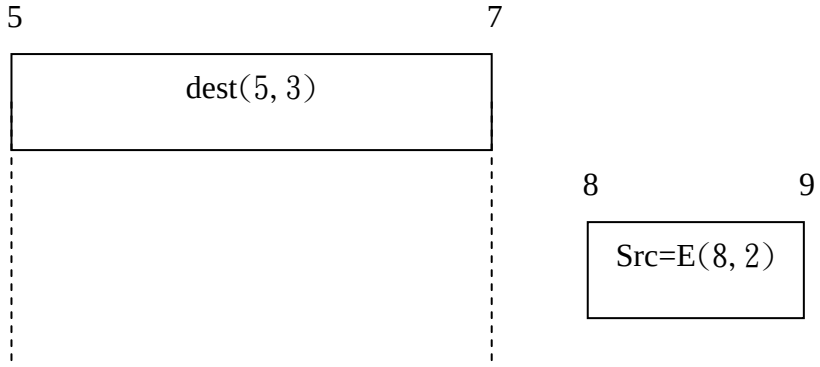

圖 3-8：RLE 模組 F 情形(無交集)

狀況二的情形為：dest_right < Src_rihgt

F 情形 =完全包含情形：dest 之編碼為(5,3)、Src 之編碼為(4,5)，

F(4,5)則與 dest 交集長度為 3，所以 dest,F 交集的部份，從位置 5 開始到 7，則以 dest,F(5,3)來表示，如圖 3-9

圖 3-9：RLE 模組 F 情形(完全包含)

3.3.2 RLE 演算法

關於 RLE 演算法虛擬碼如下圖 3-10 所示：

```
1.   L1=find_frequent_1-itemsets(D);
2.   for(K=2 ; L_{K-1} ≠ 0 ; K++) do begin
3.  {
4.  L_K=φ;
5.  C_K-itemset = apriori-gen(L_{K-1}) //產生 Ck
6.        for all candidates c in C_K-itemset do ;
7.        {
8.        count = getSupport(C_K-itemset , min_sup)
9.        if(count >= min_sup) L_K=L_K ∪{C};
10.       }
11.}
12.return L = ∪_K L_K;
```

圖 3-10 RLE 演算法虛擬碼

RLE 演算法經比對後產生高頻項目集之步驟說明

一、 將L_{k-1} 的高頻重新組合產生C_k的候選集

二、 再將C_k候選集的RLE List帶入RLE模組

三、 And_op()判斷所屬分類狀況進行交集比對

四、 getSupport()則回傳此次候選項目所比對的結果 K 頻繁項目集

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

```
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)         //case 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 3-11 RLE 演算法

3.4 RLE 演算法範例說明

這裡將以一個範例說明所提出 RLE 演算法之挖掘過程。

第一步驟：進行前置處理編碼並且產生 L1-itemset 即為 RLE

第二步驟：再由 L1 產生 C2 的候選項目集

第三步驟：將 C2 候選項目集之 RLE List Set 載入 RLE 模組 And_op() 方法中執行，將產生 L2 高頻項目集之 RLE List Set

第四步驟：重複進行第二、第三步驟，直到沒有候選項產生為止

3.4.1 RLE 演算法探勘資料說明

這個範例主要是說明如何利用二個 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 3-1 是一個交易資料庫，裡面有 10 筆交易，七種不同的項目。藉由前

述的演算法編碼，可將該資料轉為如表 3-3 的 RLE 串列。而表 3-4 包含所有產生的頻繁項目集合與其 RLE list Set 內容。

初始設定值 Support 最小支持度=3

表 3-3：交易資料之 RLE

itemset	RLE List Set	supcount
A	[(2,1),(4,3),(9,2)]	6
B	[(1,1),(3,2),(7,4)]	7
C	[(1,1),(8,1)]	2
D	[(1,1),(6,3)]	4
E	[(3,5),(10,1)]	6
F	[(2,2),(5,2),(10,1)]	5
G	[(3,1),(8,1)]	2

經對每一項目的 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-頻繁項目集如表 3-3 中的 C、G 是沒有超過 Support 值，所以該兩個 items 並不會出現在 L1 的頻繁項目集內，如表 3-4，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 3-4：1-頻繁項目集

itemset	RLE List Set	supcount
A	[(2,1),(4,3),(9,2)]	6
B	[(1,1),(3,2),(7,4)]	7
D	[(1,1),(6,3)]	4
E	[(3,5),(10,1)]	6
F	[(2,2),(5,2),(10,1)]	5

接下來由 1-頻繁項目集來產生 2-itemset 的候選項目集，如表 3-5

表 3-5： 2-候選項目集(C_2)

2-itemset 候選項目集
AB
AD
AE
AF
BD
BE
BF
DE
DF
EF

以 AB 為例，取出 A、與 B 的 RLE 串列，即為 A=[(2,1),(4,3),(9,2)]、B=[(1,1),(3,2),(7,4)]，將 A 與 B 的串列帶入 RLE 的演算法中進行 And_op()交集處理後得到 AB= (4,1),(9,2)，則表示第 4 個位置開始有 1 個長度、以及第 9 個位置開始有 2 個長度是交集的部份，故 AB 有交集的部分長度為 3，由上述等做法方可得知如表 3-6 為 2、3-高頻項目集 RLE

表 3-6： 2、3-高頻項目集 RLE

itemset	supcount
AB	3
AE	4
AF	4
BD	3
BE	3
EF	4

itemset	supcount
AEF	3

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之 RLE 作運算比對來產生高頻項目集，如更改最小支持度時，則無需重新掃瞄資料庫，只須對記憶體中的 LRE List Set 重新更新及探勘即可。

3.4.2 RLE 演算法範例示意圖

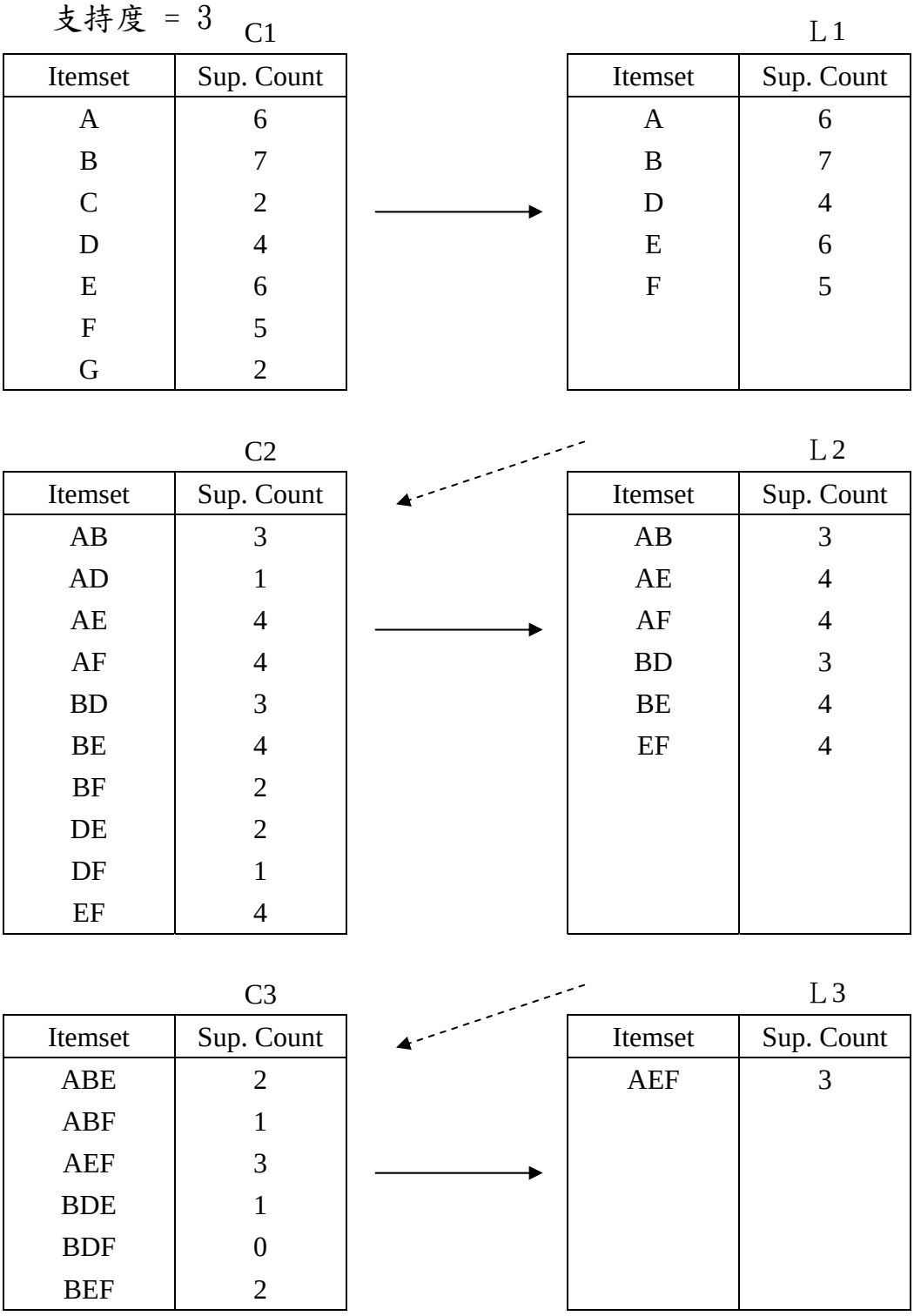

圖 3-12：RLE 演算法範例示意圖

第四章　效能分析與實驗結果

本章將對我們提出的基於連續長度編碼之關聯法則挖掘法 RLE 與相關著名的演算法用不同觀點來做效能分析。在 4.1 節中我們先描述實驗環境的建置，我們主要分成兩個部份來說明，第一部份為硬體環境，第二部份為測試資料庫之建置。在 4.2 節中做掃描資料庫的效能分析。接著 4.3 節中實作各種演算法的效能分析，第一個為分析當資料庫的交易資料量增加時，其各個演算法之效能差異，第二個為分析項目種類增加時，其各個演算法之效能差異，第三個為分析當資料庫的交易長度不同時，其效能差異。在 4.4 節中進行記憶體使用之效能分析，我們利用了三個不同方向來分析在不同編碼時，記憶體之效能分析，第一部份為分析當資料庫的交易資料量增加時，記憶體之使用量。第二部份為分析當項目種類增加時，記憶體之使用量。第三部份為分析當交易長度增加時，其記憶體之使用量，在這裡我們發現當編碼方式能夠佔用越少的記憶體，則能夠有效的加快探勘的速度。有這三種方向來分析，可讓我們了解這些演算法在不同支持度下其執行的效能。

4.1 實驗環境之建置

在實驗環境部份，我們主要分成二個部份來說明，第一部份為硬體環境，第二部份為測試資料庫之建置。

一、硬體環境

本研究利用 Microsoft Visual C# 2005 開發 基於連續長度編碼之關聯法則挖掘法，硬體平台 CPU 為 Pentium 4G、1024MB 主記憶體、80G IDE 硬碟。測試平台之 OS 作業系統為 Windows XP 專業版。對於作業系統中不必要的常駐程式在測試前皆已關閉。所有應用程式和服務程式也都結束，只單純使用本 RLE 演算法來執行。

二、測試資料庫

在建立測試資料方面，均採用 IBM Almaden Research Center[4]所開發的資料產生器所產生的資料作為測試資料。 我們依據 Apriori[8]文獻中所提到的方法來產生交易資料庫。此類的資料庫經被用來當作測試資料用，比較有公信力。資料在產生時有幾個參數可供設定，我們將產生表 4-1 資料庫的參數列於下表。

表 4-1：資料庫參數設定

Ntrans	Tlen	Patlen	nitems	備註
D	T	I	N	Ta.Ib.Nc.Db
原始資料庫中交易的數量	每筆交易中的平均長度	可能是高頻項目組集合數量的平均值	項目的數量	例如：T10I4N1KD100K 表示此資料庫有 10 萬筆資料，其中每筆交易為 10 筆，高頻項目組平均為 4 個，項目種類為 1000 個

4.2 掃描資料庫的效能分析

當在進行資料探勘階段，不同演算法其掃瞄資料庫的次數也不同。大多數的探勘關聯演算法之文獻中，皆在討論掃描資料庫的次數，作者們也都提出減少掃瞄資料庫的次數，將可有效提升探勘的效能，故在本分析中，我們對著名的演算法進行分析。我們所提出的演算法 RLE，由於只掃瞄 RLE List Set 而不需掃描資料庫，故可以加快資料探勘的速度。表 4-2 為各個演算法掃瞄資料庫之比較

表 4-2：各個演算法掃瞄資料庫次數之比較

	掃瞄資料庫的次數
Apriori	每找尋一次高頻項目組，皆需要掃瞄一次
FP-tree	掃瞄資料庫二次
DLG	掃瞄資料庫一次
RLE	掃瞄資料庫一次，後續只需掃瞄 RLE List Set

4.3 各種演算法之效能分析

為了分析在不同支持度下，不同演算法之效能，我們從三個方向進行分析，第一個為分析當資料庫的「交易資料量增加時」，其各個演算法之效能差異。

第二個為分析當「項目種類增加時」，其各個演算法之效能差異。

第三個為分析當「交易長度增加時」，其各個演算法之效能差異。

以下將詳細說明此三項分析。

一、當資料庫的交易資料量增加時：

本實驗中，我們利用目前最快的關聯規則演算法 FP Growth 與我們的基於連續長度編碼之關聯法則挖掘法在各種不同的最小支持度作比較。利用交易資料量的增加，對於各個演算法執行效能的影響。表 4-3 為本實驗所使用的資料庫參數設定，這四個資料庫的項目數量均為 1000、平均交易長度均為 10、平均高頻項目組集合長度均為 4，而交易資料量分別為 1 萬、3 萬、5 萬、10 萬筆。

表 4-3：只增加交易資料量之資料庫

D	T	I	N
10000	10	4	1000
30000	10	4	1000
50000	10	4	1000
100000	10	4	1000
T10I4N1KD10K			
T10I4N1KD30K			
T10I4N1KD50K			
T10I4N1KD100K			

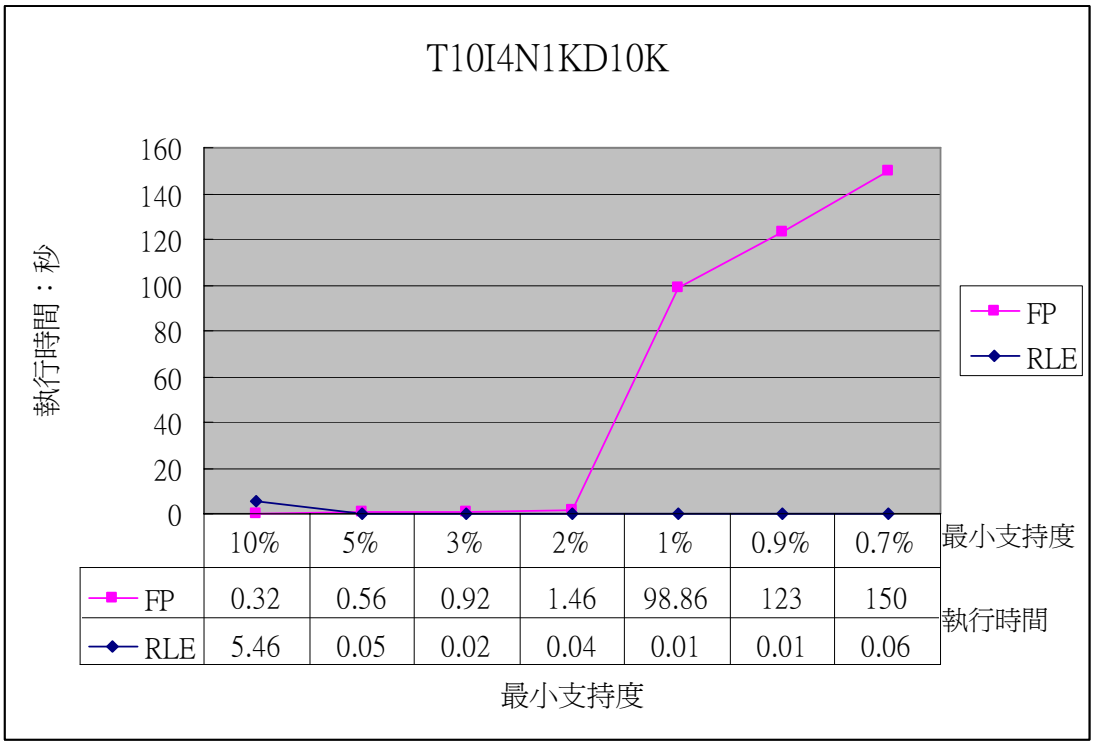

圖 4-1：資料量為 10000 筆時之 RLE 與 FP Growth 演算法之效能分析

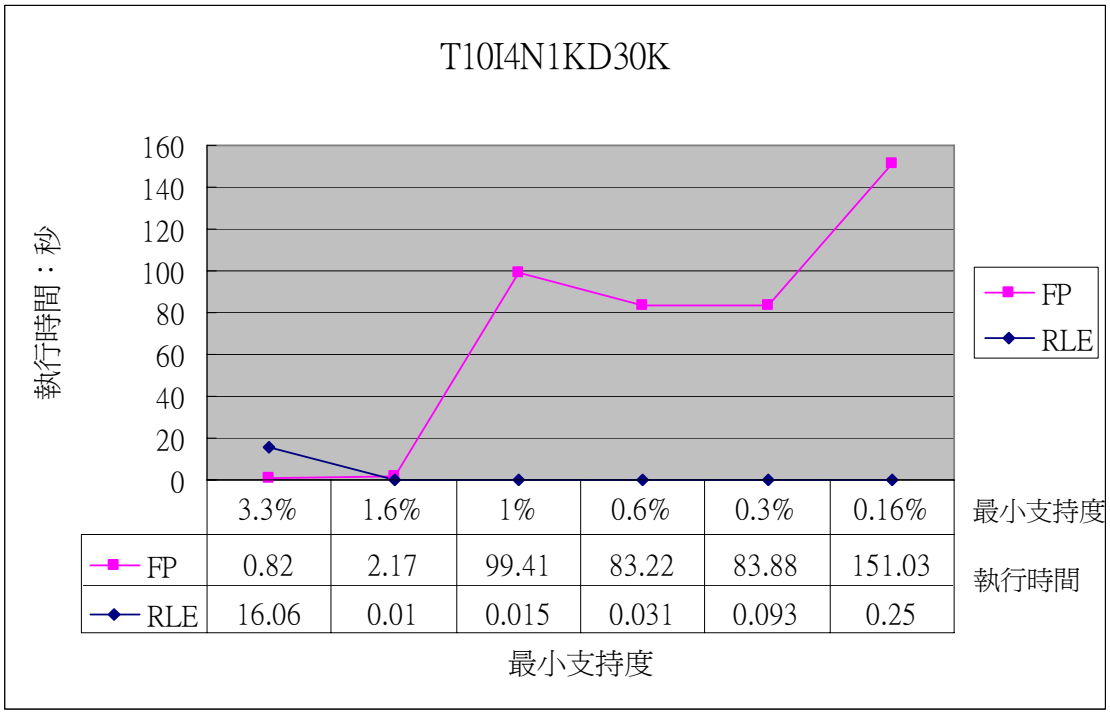

圖 4-2：資料量為 30000 筆時之 RLE 與 FP Growth 演算法之效能分析

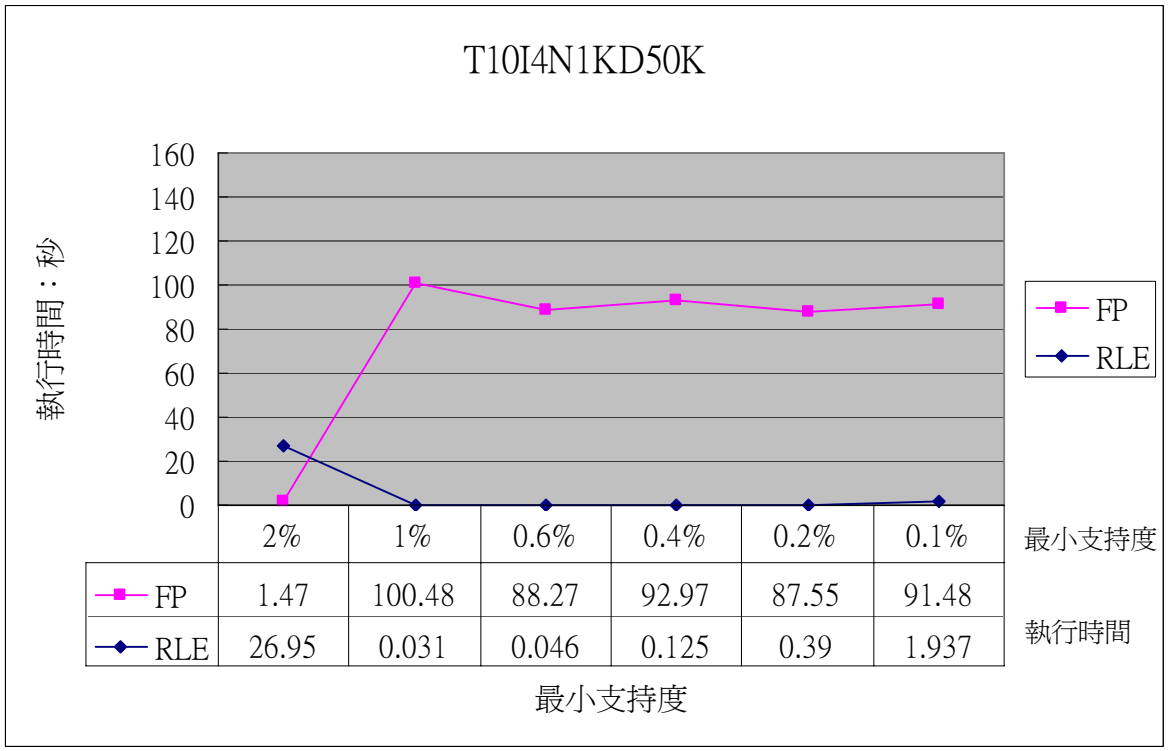

圖 4-3：資料量為 50000 筆時之 RLE 與 FP Growth 演算法之效能分析

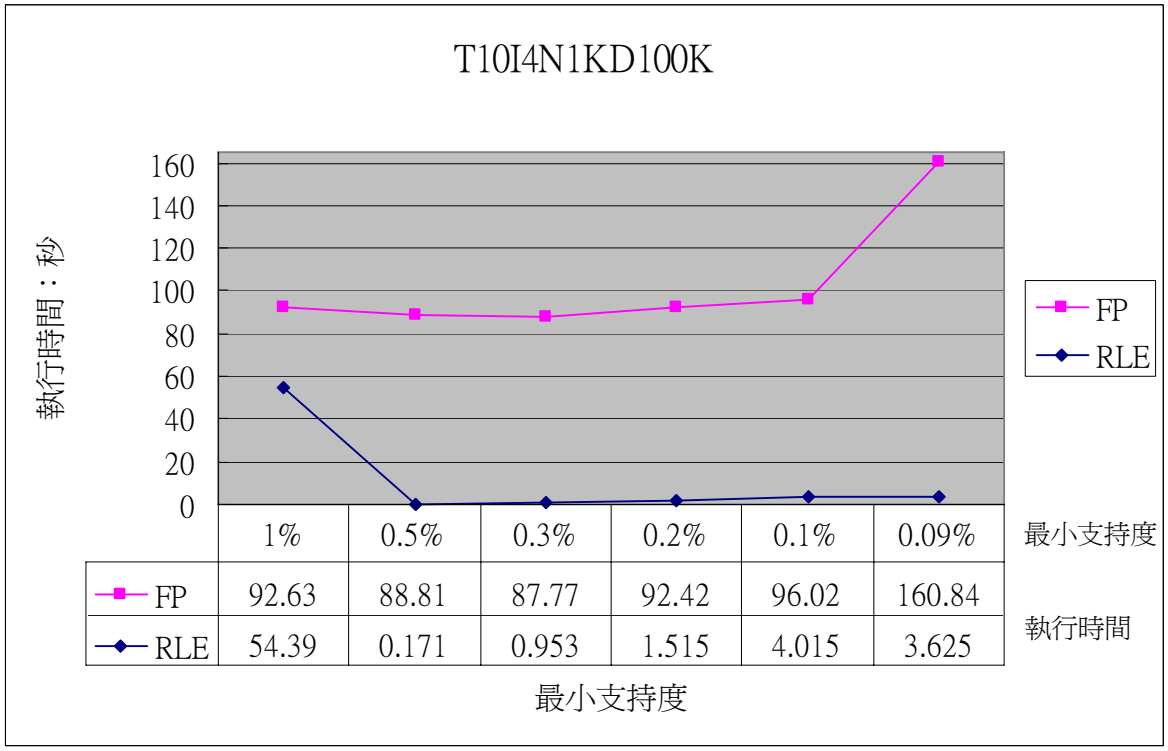

圖 4-4：資料量為 100000 筆時之 RLE 與 FP Growth 演算法之效能分析

由以上實驗結果，我們可以觀察到當資料量越大時，則 RLE 演算法將比 FP Growth 的探勘速度更加快速，尤其支持度在 1%以下，幾乎以接近 0 秒的時間就可以找出所要的資訊，因此證明此方法其可以達到有效率的進行探勘關連規則。

二、當資料庫的項目種類增加時：

本實驗中，我們利用目前最快的關聯規則演算法 FP Growth 與我們的基於連續長度編碼之關聯法則挖掘法 RLE 演算法在各種不同的最小支持度作比較。利用項目種類數量的增加，對於各個演算法執行效能的影響。表 4-4 為本實驗所使用的資料庫參數設定，這二個資料庫的交易資料量為 10000 筆、平均交易長度均為 10、平均高頻項目組集合長度均為 4，而項目數量分別為 100、500、1000 筆。

表 4-4：只增加項目種類數量之資料庫

D	T	I	N
10000	10	4	100
10000	10	4	500
10000	10	4	1000
T10I4N100D10K			
T10I4N500D10K			
T10I4N1KD10K			

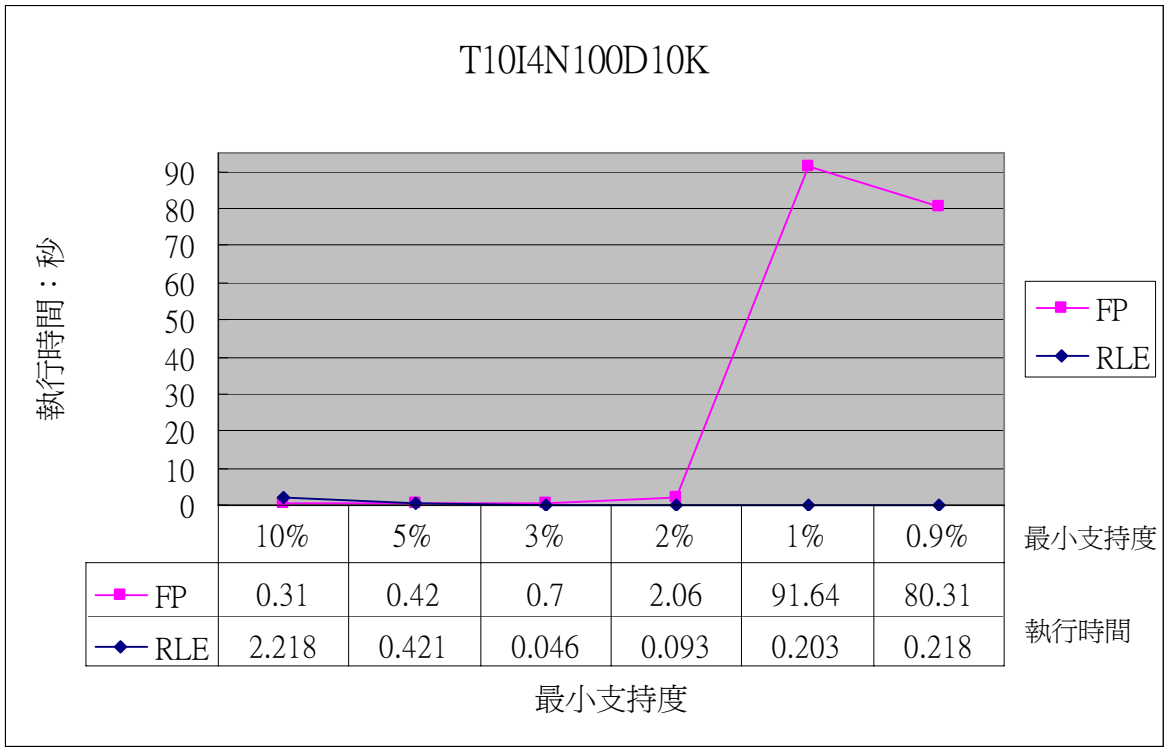

圖 4-5：項目數量為 100 筆時之 RLE 與 FP Growth 演算法之效能分析

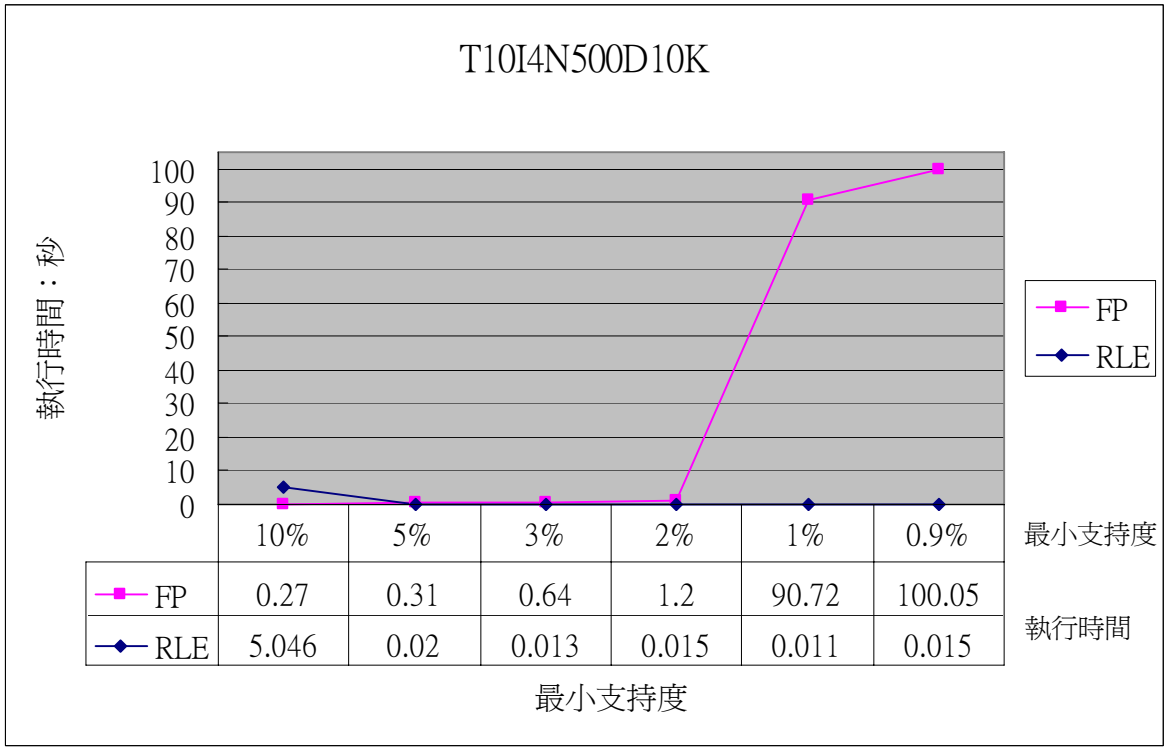

圖 4-6：項目數量為 500 筆時之 RLE 與 FP Growth 演算法之效能分析

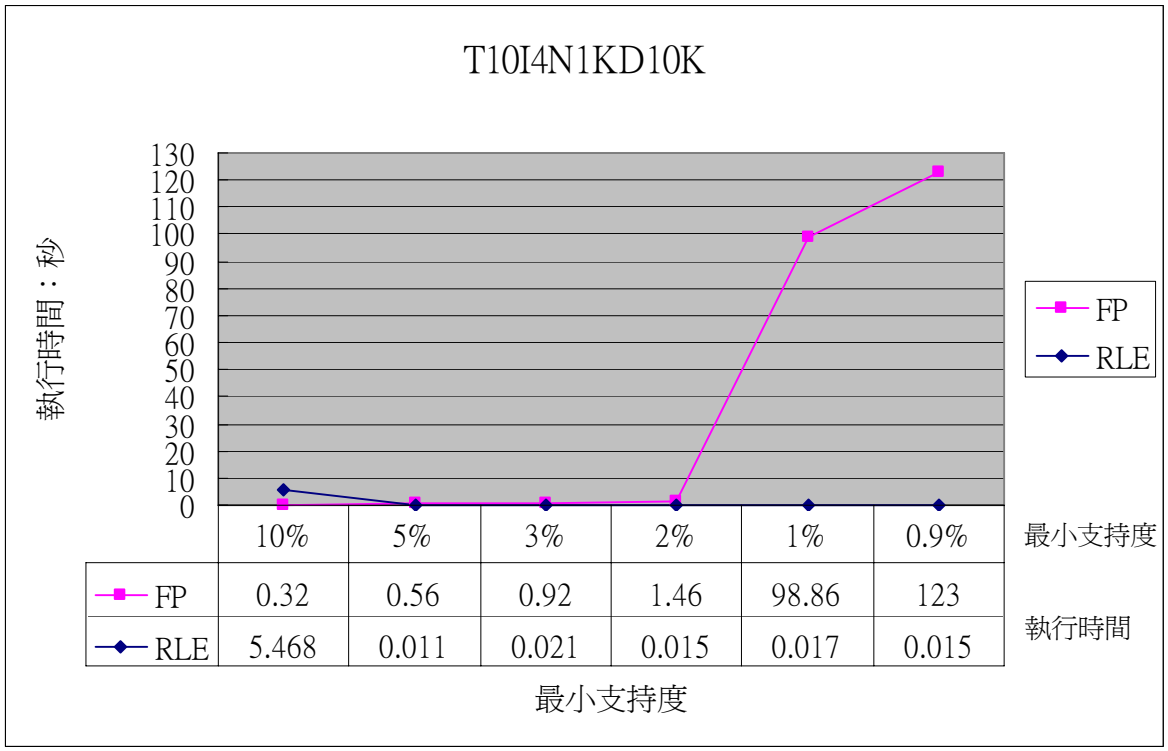

圖 4-7：項目數量為 1000 筆時之 RLE 與 FP Growth 演算法之效能分析

由以上實驗結果，我們可以觀察到當項目種類越多時，則 RLE 演算法將比 FP Growth 的探勘速度更加快速。

三、當資料庫的交易長度增加時：

本實驗中，我們利用當資料庫中的交易長度不同時，對於各個演算法執行效能的影響。表 4-5 為本實驗所使用的資料庫參數設定，這三個資料庫的項目數量均為 1000 個、交易資料量均為 1 萬筆，而平均高頻項目組長度為 4、平均每筆交易長度為 3、5、10。

表 4-5：只增加交易長度之資料庫

D	T	I	N
10000	3	4	1000
10000	5	4	1000
10000	10	4	1000
T3I4N1KD10K			
T5I4N1KD10K			
T10I4N1KD10K			

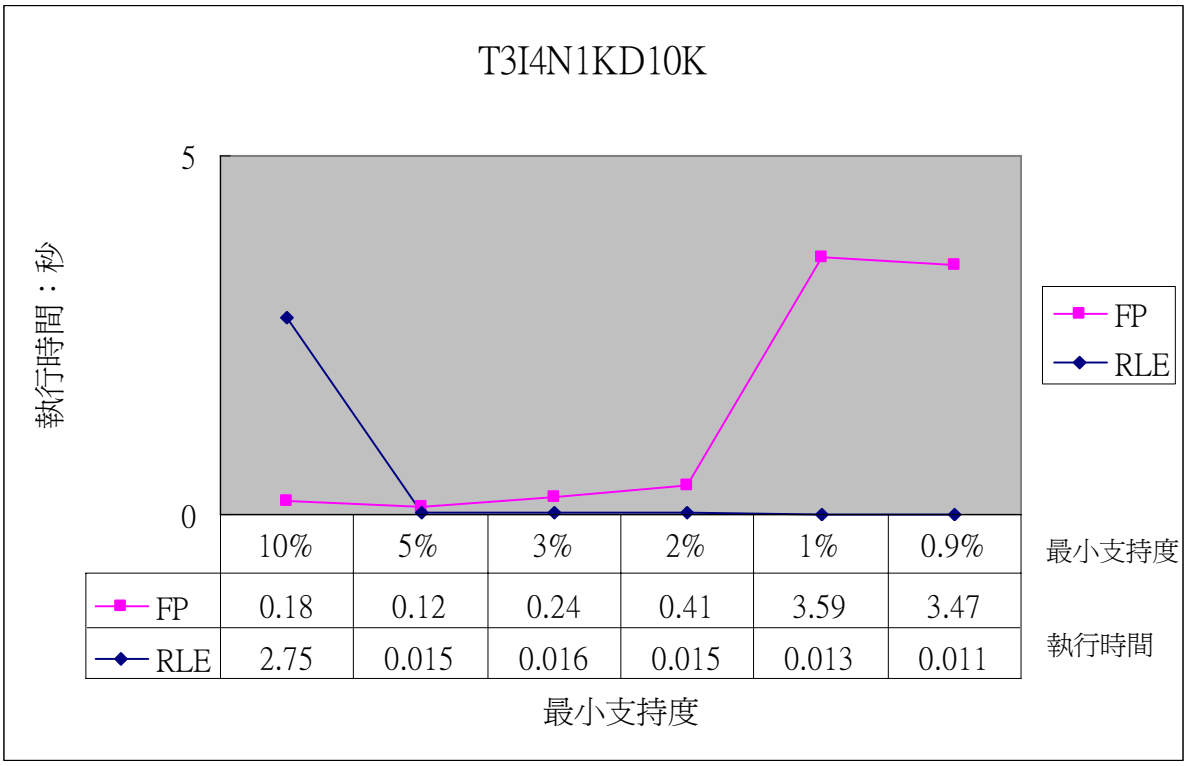

圖 4-8：平均交易長度為 3 筆時之 RLE 與 FP Growth 演算法之效能分析

圖 4-9：平均交易長度為 5 筆時之 RLE 與 FP Growth 演算法之效能分析

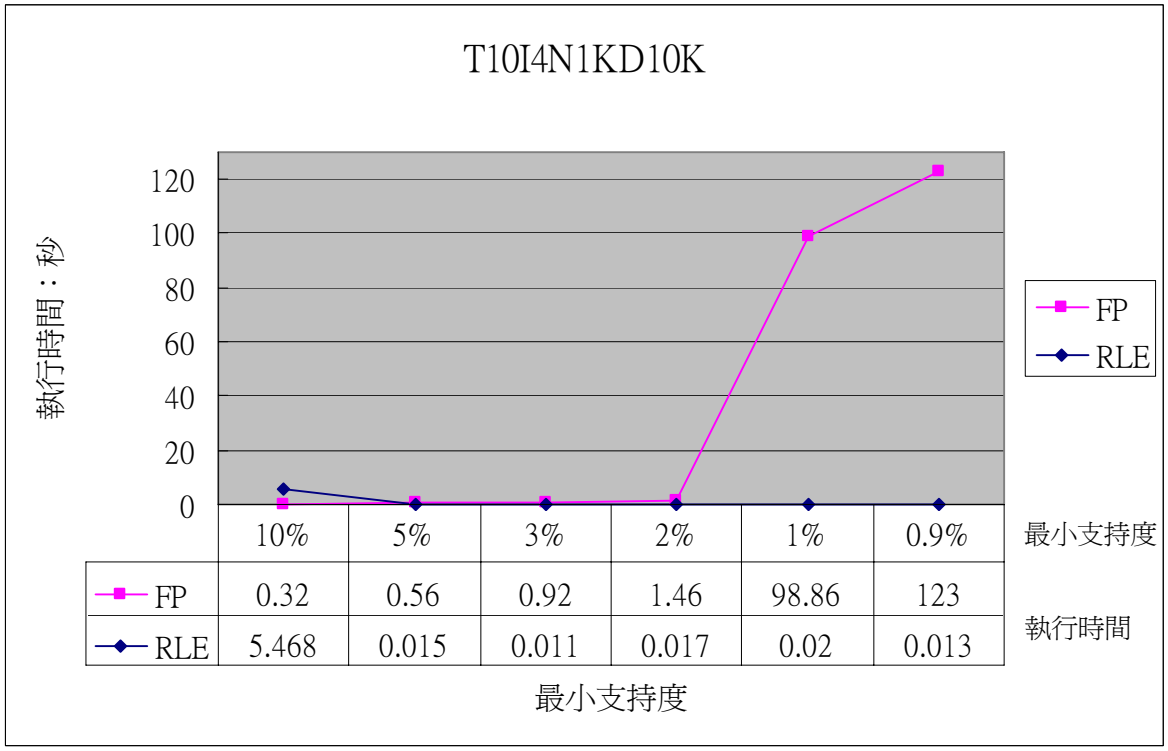

圖 4-10：平均交易長度為 10 筆時之 RLE 與 FP Growth 演算法之效能分析

由實驗的結果可以得知當資料庫越稀疏時，利用基於連續長度編碼之關聯法則挖掘法 RLE 演算法可以有效率的壓縮資料，使得探勘效能越佳。

4.4 各種演算法記憶體使用之效能分析

在本實驗中將分析不同編碼方式下，對於支持度計算的效能比較，我們將比較 Bit-Vector 與我們提出的 RLE 在效能上的差異。當以 Bit-Vector 編碼方式進行資料探勘時，若資料庫中的交易越多時，則每個項目的位元向量就會越長。當資料庫中的資料量變大，或是資料庫中的項目變多時，Bit-Vector 所佔用的記憶體空間將會非常大。為了解決這個缺點，我們提出 RLE 編碼方式，其可以有效的達到減少所使用的記憶體空間。

為了分在不同編碼方式下其記憶體使用之情況，我們對 Bit-Vector 與我們所提出的 RLE 二種編碼方式進行比較。我們從三個方向進行分析，第一部份為分析當資料庫的交易資料量增加時，不同編碼方式下，其記憶體的使用量。第二部份為分析當項目種類增加時，不同編碼方式下，其記憶體的使用量。第三部份為分析當交易長度增加時，不同編碼方式下，其記憶體的使用量。以下將詳細描述此三項分析。

一、 記憶體使用之實驗 – 增加交易筆數

在這個實驗中，我們改變資料庫的交易數量，從 1 萬筆一直增加到 10 萬，如表 4-6，藉由不同的交易數量來分析比較使用 Bit-Vector 與 RLE 兩種編碼方式下，其記憶體佔用的空間。

表 4-6：只增加交易資料量之資料庫

D	T	I	N
10000	10	4	1000
30000	10	4	1000
50000	10	4	1000
100000	10	4	1000
T10I4N1KD10K			
T10I4N1KD30K			
T10I4N1KD50K			
T10I4N1KD100K			

由圖 4-11 得知，使用 RLE 的儲存方式所佔用的空間比用 Bit-Vector 的減少很多，尤其在越大型的資料庫中，其壓縮效果更是顯著。我們以一個壓縮率的公式，如公式 4-1 來表示利用 RLE 壓縮 Bit-Vector 所產生的效益，當壓縮率的值越大時，表示利用 RLE 的壓縮方式越能達到有效壓縮。

令 B=Bit-Vector 在記憶體中所占用的空間

R=RLE 在記憶體中所占用的空間

$$\text{壓縮率} = \frac{B - R}{B} \quad (\text{公式 4-1})$$

表 4-7：利用 RLE 編碼方式之壓縮效率

壓縮率	
T10I4N1KD10K	86.52 %
T10I4N1KD30K	86.74 %
T10I4N1KD50K	87.31 %
T10I4N1KD100K	87.65 %

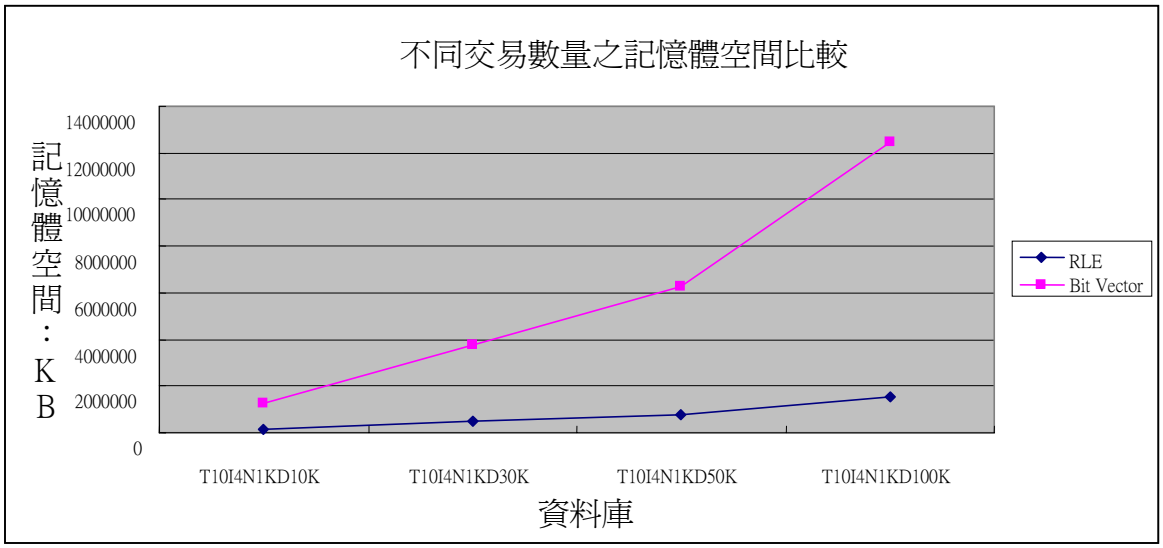

圖 4-11：不同交易數量之記憶體空間比較圖

二、 記憶體使用之實驗 – 增加項目種類數量

在這個實驗中，我們改變每個資料庫的項目數量，從項目為 100 個項目到 1000 個項目，其他參數均相同如表 4-8，藉由不同的項目數量來分析比較使用 Bit-Vector 與 RLE 兩種不同儲存資料庫的方法，在記憶體佔用的空間。

表 4-8：只增加項目種類數量之資料庫

D	T	I	N
10000	10	4	100
10000	10	4	500
10000	10	4	1000
T10I4N100D10K			
T10I4N500D10K			
T10I4N1KD10K			

由圖 4-12 可知，使用 RLE 的儲存方式所佔用的空間比用 Bit-Vector 的減少很多。

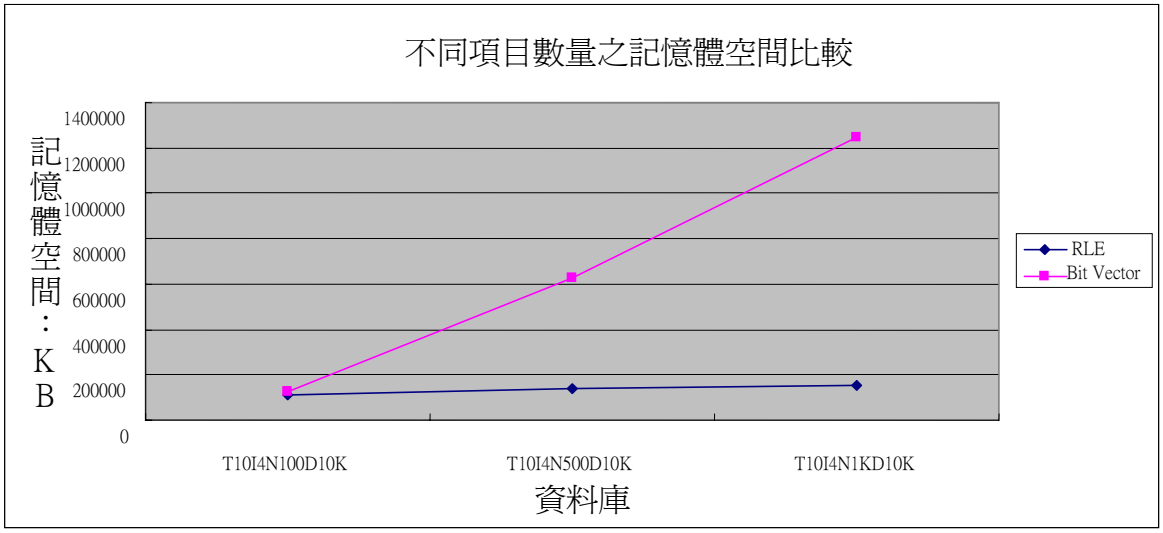

圖 4-12：不同項目數量之記憶體空間比較圖

表 4-9：利用 RLE 編碼方式之壓縮效率

壓縮率	
T10I4N100D10K	12.54 %
T10I4N500D10K	77.71 %
T10I4N1KD10K	87.64 %

三、 記憶體使用之實驗 – 增加交易長度

在這個實驗中，我們改變每個資料庫的交易長度，從交易長度為平均 3 筆到平均 10 筆，其他的參數均相同，如表 4-10，藉由不同的項目數量來分析比較使用 Bit-Vector 與 RLE 兩種不同儲存資料庫的方法，在記憶體佔用的空間。

表 4-10：只增加交易長度之資料庫

D	T	I	N
10000	3	4	1000
10000	5	4	1000
10000	10	4	1000
T3I4N1KD10K			
T5I4N1KD10K			
T10I4N1KD10K			

由圖 4-13 可知，使用 RLE 的儲存方式所佔用的空間比用 Bit-Vector 的減少很多空間。在表 4-11 中壓縮率表現，可知在交易長度越長時，壓縮率有降低的趨勢，是因為交易長度的增加，使用項目出現的頻率也隨著增加，故造成壓縮率降低之情形。

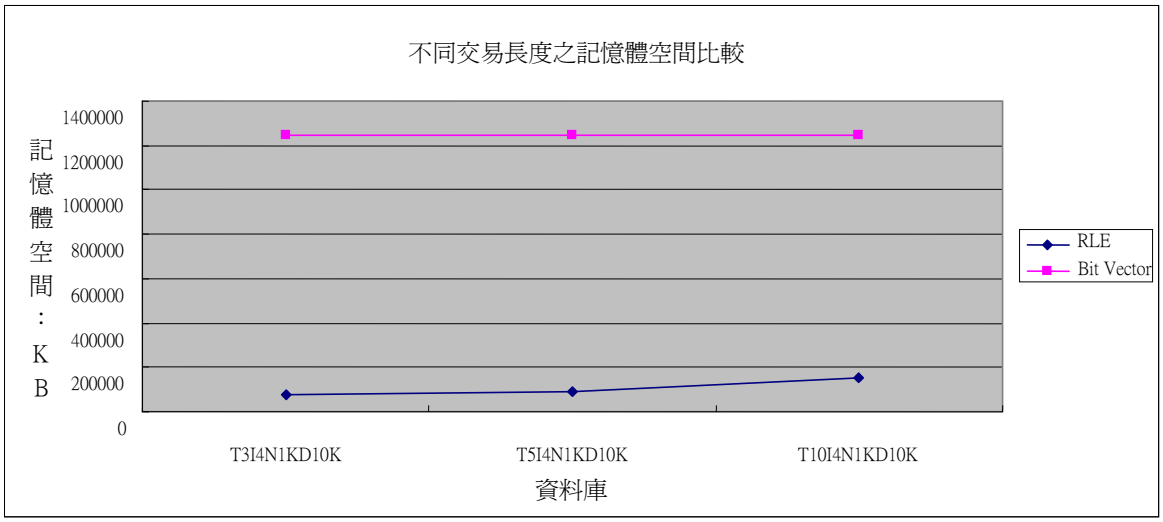

圖 4-13：不同交易長度之記憶體空間比較圖

表 4-11：利用 RLE 編碼方式之壓縮效率

壓縮率	
T3I4N1KD10K	94.1 %
T5I4N1KD10K	92.5 %
T10I4N1KD10K	87.64 %

綜合以上實驗結果分析，我們提出的基於連續長度編碼之關聯法則挖掘法 RLE 演算法，能優於 FP Growth 原因如下：

利用前置處理建置好 RLE List Set 後，就利用我們提出的 RLE 演算法，在不用解壓縮情形下即可直接做資料探勘。當改變最小支持度時，也不需再重新掃瞄資料庫，也不需要重新建立新的 RLE List Set。反觀 Apriori 演算法與 FP Growth 演算法，當改變最小支持度時，必須重新掃瞄資料庫，FP Growth 更要花時間重新建置 FP-tree 才能做資料探勘，尤其在越大型的資料庫上，FP Growth 花費在建置 tree 的時間還要很多，這樣重複的再建立 tree 是很花費成本與時間。

第五章　結論與未來發展

論文中我們提出基於連續長度編碼之關聯法則挖掘法 Run-Length Encoding(RLE)，此方法主要利用以連續長度編碼方法，將交易資料庫壓縮成少量的編碼資料，進而在主記憶體中對編碼後的交易資料進行資料探勘，在探勘過程中不需要再另外進行解碼的動作，因此可以減少許多動作，進而提升探勘效率，亦解決了許多演算法遇到資料庫資料量太大時，還必須將部份的交易資料存到磁碟機內，而造成資料探勘速度降低的問題所在。

此外，我們只需要掃瞄交易資料庫一次，即完成 RLE List Set 的建置程序，無需在第二次掃瞄的動作，不論後續進行幾次不同支持度的探勘，皆不需要重覆執行掃瞄動作，如此亦可增加探勘上的效率。

經分析、與實驗證實，我們所提出的 RLE 方法可以大量減少不必要的處理動作，並且也在多次的探勘實驗下，RLE 方法優於 FP-Growth 演算法。

未來發展

在這部份目前則是以單機應用程式來撰寫程式及探勘資料，將來可以發展到網路線上直接探勘資料，即克服了空間的問題，只要在有網路設備的地方就能上網進行探勘資料，如此就能讓使用者更方便的取得所想要的資訊，進而做為企業產品關聯之組合策略。

參考文獻

[1] S. J. Yen and A. L. P. Chen. "An Efficient Approach to Discovering Knowledge from Large Databases." Proc. IEEE/ACM International Conference on Parallel and Distributed Information Systems (PDIS), (1996).

[2] Han, J., Pei J. and Yin Y., "Mining Frequent Patterns without Candidate Generation" Proceedings of ACM-SIGMOD Interna-tional Conference on Management of Data, pages 1-12, May (2000).

[3] J. Pei, J. Han. H. Lu, S. Nishio, S. Tang, and D. Yang. "H-Mine: Hyper-structure Mining of Frequent Patterns in Large Databases" Proc. The 2001 IEEE International Confer-ence On Data Mining (ICDM' 01), San Jose, California, November 29-December 2, (2001).

[4] IBM Data Mining Web Site: http://www.almaden.ibm.com/cs/quest/index.html

[5] M. S. Chen, J. Han, and P.S. Yu, "Data Mining: An Overview from a database Perspective", IEEE Transaction on Knowledge and Data Engineering, Vol. 8,No. 6,December (1996).

[6] R. Agrawal and R. Srikant, "Fast Algorithm for Mining Assoication Rule in Large Databases", In Proc. 1994 Int'l Conf. VLDB, pp. 487-499, Santiago, Chile, Sep. (1994).

[7] Savasere, E. Omiecinski, and S. Navathe, "An Efficient Algorithm for Mining Association Rule in Large Database", Proc. 21th VLDB,pp. 432-444, (1995).

[8] R. Agrawal, T. Imielinski, & A. Swami, "Mining Association RuSets of Items in Large Database," Proceedings of SIGMOD, USA, pp. 207-216 (1993).

[9] R. Agrawal, T. Imielinski and A. Swami, "Database Mining: A Perspective," IEEE Transactions on Knowledge and Data Engi914-925 (1993).

[10] R. Agrawal, C. Faloutsos and A. Swami, "Efficient SimilaritSequence Databases," Lecture Notes in Computer Science 73Verlag, pp. 69-84 (1993).

[11] 邱士軍，"關聯規則演算法之實作和效能評估"，國立台灣科技大學資訊管理研究所碩士論文(2002)。

[12] 徐雅琪，"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(2003)。