從客戶交易資料庫中發掘高效益序列之研究

顏秀珍
銘傳大學資訊工程學系
sjyen@mail.mcu.edu.tw

李御璽
銘傳大學資訊工程學系
leeys@mail.mcu.edu.tw

顧家源
銘傳大學資訊工程學系
496479451@ntnu.edu.tw

摘要

高效益序列型樣探勘(High Utility Sequential Pattern Mining)主要是從交易資料庫中找出能帶來最高獲利的商品購買序列。不同於傳統的序列型樣探勘，我們的研究多考慮了交易中商品被購買的數量與每樣商品的單位利潤，並將評估一個序列是否重要的指標由原本的出現頻率改為帶來的效益總值。因此，本篇論文提出一種新的計算序列效益的方式，不但能使過去計算效益的方式加以簡化，也能確保找出來的高效益序列型樣是具有應用價值的；同時，也提出一套新的探勘方式，不但能更進一步縮減搜尋空間，還可以大幅降低序列效益的高估情形，使不必要的運算被降至最低。

關鍵字：資料探勘、高效益序列型樣、交易資料庫

緒論(Introduction)

在這個資訊科技發達的社會裡，取代傳統人力記帳的方式，使用電腦來記錄龐大的交易資料已是相當普遍的事；隨著電腦的硬體設備不斷發展演進，執行效能與儲存空間不斷增加，而儲存的交易資料量也是與日俱增。然而，這些儲存之交易資料中常常隱藏著一些重要的商業知識，但龐大的資料量卻讓這些知識難以被發現；因此，如何快速的從大量資料中找出有用的知識便成了一個重要的課題，也從而誕生了「資料探勘」或「資料採礦」一詞。

而在資料探勘的領域中，序列型樣探勘（Sequential Pattern Mining）是一個相當重要的研究。所謂的序列型樣，便是一連串有時間順序性且具有相當代表性的事件。例如在交易資料庫中，可能發現大部分的顧客購買了A商品後，通常會再購買B商品；找出這樣的序列型樣，可利於預測顧客下次購買之需求，並針對其需求進行優惠或推薦，藉此達成顧客保留或交叉銷售等目的。

到目前為止，序列型樣探勘之研究大多將其目標放在找出資料庫中「多數消費者」所共通具有的有時間順序之購買行為，例如在所有顧客的交易資料中，發現有一定比例的顧客曾經發生過買完A之後買B的行為，則認為這樣的行為模式是具有代表性的；不過，僅考慮商品「是否」被購買，卻忽略了某些其他重要的因素，例如顧客購買商品的數量與售價等，如此便可能只找到雖然常常被循序購買，但對商家帶來的總利潤卻不高的商品組合，而無法發現雖然出售的次數較少，但卻帶給商家更多利潤的商品組合；然而對於商家來說，在有限的行銷資源下要能有效的提升獲利，這種高利潤的商品組合才往往是行銷的重點。因此，本篇論文將提出一個高效益序列型樣探勘（High Utility Sequential Pattern Mining）的研究。對於高效益序列型樣探勘，我們做了以下的定義：

一個商品即為一個*項目（item）*，一個*項目集（itemset）*是項目的集合，表示為 $\{i_1, i_2, \ldots, i_k\}$，其中 i_j（$1 \le j \le k$）為一個項目；而一筆*交易（transaction）*某一顧客同時購買所有項目的集合。*序列（sequence）* $S = < s_1\ s_2\ \cdots\ s_n >$中，$s_i$稱為序列 S 的一個*元素（element）*，且一個元素代表一個項目集，表示一連串有順序性的商品購買行為；例如$< X_1\ X_2\ X_3 >$就是代表顧客先買了 X_1，之後買了 X_2，最後又買了 X_3 的行為。序列 S 的*長度(length)*為 S 中項目的個數；例如序列$< \{a\ b\}\ a\ c\ d >$為長度為 5 的序列，亦可表記為 5-序列。序列 S 中項目 i_p 的*位置（location）*則記為 $loc(i_p, S) = x$，表示 i_p 出現在序列 S 的第 x 個項目集中，例如項目 c 在序列 $S = < \{a\ b\}\ a\ c\ d >$中的位置 $loc(a, S) = \{1, 2\}$。一個序列 $S' = < s'_1\ s'_2\ \cdots\ s'_k >$為序列 S 的*子序列(subsequence)*，則 $s'_1 \subseteq s_{i1}, s'_2 \subseteq s_{i2}, \ldots, s'_k \subseteq s_{ik}$（$1 \le i_1 < i_2 < \cdots < i_k \le n$），而 S 則為 S'的*超序列（supersequence）*；序列 S 中的一個*區段(segment)*記為 $SG([l_1, l_2], S) = < s_{l1}\ s_{l1+1}\ \cdots\ s_{l2} >$（$l_1 \le l_2$），其中 $loc(s_i, S) = i$（$l_1 \le i \le l_2$），表示序列 S 中由第 l_1 個項目集到第 l_2 個項目集中所有項目集所構成的 S 的子序列，例如 $S = < \{a\ b\}\ a\ c\ d >$，則 $SG([1, 3], S) = < \{a\ b\}\ a >$。一個*使用者交易序列（user-transaction-sequence）*為某一顧客的所有交易依照購買時間先後順序排列所形成的序列，除了記錄此顧客依序先後購買了哪些商品，同時也包含了每個被購買的商品在

該次交易中被購買的*數量（Quantity）*，而一個使用者交易序列所對應的*使用者序列（user-sequence）*則是不含購買數量的資訊；例如使用者交易序列 US^T = < a(2) { b(3) c(1) } >表示此顧客先購買了 2 個商品 a，下次交易又同時購買了 3 個商品 b 與 1 個商品 c，而對應 UTS 的使用者序列 US = < a {b c} >。項目 i_p 的*單位利潤（profit）*表示為 prof(i_p)，也就是每販售出一個商品 i_p 可得到的獲利；項目的*單位利潤表（profit table）*則記錄了資料庫中每一個項目的單位利潤，如表 1 所示。一個*序列資料庫（sequence database）* SDB 為一個包含已整理為序列形式之交易紀錄的資料庫，每筆紀錄包含一個*序列編號（SID）*與對應的使用者交易序列，且 US^T_k 表示 SID 為 k 的使用者交易序列，如表 2 所示。su(i_p, US^T_q)為項目 i_p 在使用者交易序列 US^T_q 中的*效益（utility）*，其定義為項目 i_p 的單位利潤與使用者交易序列 US^T_q 中項目 i_p 的總購買數量之乘積；例如表 2 中，項目 a 在序列編號為 10 的使用者交易序列 US^T_{10} 中的效益為 $su(a, US^T_{10}) = 5 \times (3 + 2 + 5) = 50$。一個使用者序列 US_q 的效益 su(US_q)的定義如式(1)所示，

$$su(US_q) = su(US^T_q) = \sum_{i_j \in US_q} su(i_j, US^T_q) \tag{1}$$

表示 US_q 總共帶來多少利潤；例如表 2 中，$su(US_{10}) = su(a, US_{10}) + su(b, US_{10}) + su(d, US_{10}) + su(f, US_{10}) = 130$。su(SDB)為序列資料庫 SDB 的總效益，其定義如式(2)所示。

$$su(SDB) = \sum_{US_i \in SDB} su(US_i) \tag{2}$$

如表 2 中，$su(SDB) = su(US_{10}) + su(US_{20}) + su(US_{30}) + su(US_{40}) + su(US_{50}) + su(US_{60}) = 743$。

表 1 項目的單位利潤表（profit table）

Item	Profit
a	5
b	7
c	3
d	10
e	6
f	8
g	9

表 2 序列資料庫（sequence database）

SID	User-Sequence
10	< a(3) {a(2) b(6) d(2)} f(1) a(5) d(1) >
20	< e(3) {a(2) b(5)} d(1) c(4) >

30	< {c(1) f(2)} b(3) {d(1) e(4)} >
40	< a(2) {b(7) d(4)} {a(6) b(3)} e(5) >
50	< {d(1) f(3)} c(5) g(2) >
60	< d(2) e(1) {a(7) b(8)} d(3) b(6) e(3) >

為了找出某個序列 S 在序列資料庫中總共提供了多少效益，我們必須先確定每一個使用者序列 US_k 真正發生序列 S 此行為的位置，以及序列 S 在此使用者序列中所貢獻出的效益，因此我們提出以下在別的研究中未提及的定義：給定一個序列 $X = < x_1\ x_2 \cdots\ x_m >$與一個使用者交易序列 $US^T = < s_1\ s_2 \cdots\ s_n >$，若 X 為 US^T 所對應的使用者序列 US 的子序列且 $OC = < (x_1, l_1), (x_2, l_2), \ldots, (x_m, l_m) >$，其中 $l_i = loc(x_i, US)$ （$1 \le i \le m$），則我們稱 OC 為 X 在 US 中的一個*發生（occurrence）*。若 US 中存在 X 的兩個發生，分別為 $OC_1 = < (x_1, l_{i1}), (x_2, l_{i2}), \ldots, (x_m, l_{im}) >$與 $OC_2 = < (x_1, l_{j1}), (x_2, l_{i2}), \ldots, (x_m, l_{jm}) >$，且不存在任何整數值 k($1 \le k \le m$)使 $l_{ik} = l_{jk}$，則我們稱 OC_1 與 OC_2 這兩個發生彼此*互斥(disjoint)*。若 $MO = < (x_1, l_1), (x_2, l_2), \ldots, (x_m, l_m) >$為 X 在 US 的一個發生，且在 US 中的區段 SG ($[l_1, l_m\text{-}1]$, US)與 SG ($[l_1\text{+}1, l_m]$, US)中不存在任何 X 的發生，則我們稱 Y 為 X 在 US 中的一個*最小發生（minimal occurrence）*。若 $OCS = \{MO_1, MO_2, \cdots, MO_q\}$（$q \ge 1$）為 X 在 US 中之最小發生的集合且 $MO_1, MO_2, \cdots, MO_q$ 兩兩之間彼此互斥，或 OCS 中只有一個最小發生，則我們稱 OCS 為 X 在 US 中的*互斥最小發生集合（disjoint minimal occurrence set）*。若 X 在 US 中的一個互斥最小發生集合 OCS 不為任何其他 X 在 US 中之互斥最小發生集合的子集合，則我們稱 OCS 為 X 在 US 中的一個*購買行為(Purchasing Behavior)*，簡稱為 PB；序列 X 在一個使用者序列 US 中可能具有多種不同的購買行為 PB，但每個 PB 均有可能代表 X 在 US 中實際發生的購買情形。

例如令序列 X = < a d >，則表 2 中 SID 為 10 的使用者交易序列 US^T_{10} 所謂對應的使用者序列 US 中可發現 $OC_1 = < (a, 1) (d, 2) >$、$OC_2 = < (a, 1) (d, 5) >$、$OC_3 = < (a, 2) (d, 5) >$、$OC_4 = < (a, 4) (d, 5) >$均為 X 的發生，其中 OC_1 與 OC_2、OC_2 與 OC_3、OC_3 與 OC_4 均在 US_{10} 中相同的位置有相同的項目，彼此不為互斥；若之後在計算序列 X 的效益時選擇了不為互斥的發生，那便會導致某些項目的效益被重複計算。另外，OC_2 與 OC_3 均不是最小發生，因為在 US^T_{10} 中的區段 $SG([2, 5], US^T_{10}) = < \{a(2)\ b(6)\ d(2)\}\ f(1)\ a(5)\ d(1) >$與 $SG([3, 5], US^T_{10}) = < f(1)\ a(5)\ d(1) >$可找到 OC_4；與 OC_3 相較之下，OC_4 中的項目 a 和 d 被購買的時間距離較近、關係較為密切，也就是買 a 之後買 d 的行為應該是發生在 OC_4 而不是 OC_3。因為{OC_1}、{OC_4}與{OC_1, OC_4}為互斥最小發生集合，而{OC_1}與{OC_4}均為{OC_1, OC_4}的子集合，所以在 US_{10} 中{OC_1, OC_4}為 X 的購買行為。

藉由找到序列 X 在使用者交易序列 US^T_k 中的購買行為 PB，我們便可以計算序列 X 在 US^T_k 中的效益。給定一個序列 $X = < x_1\ x_2 \cdots x_m >$與一個使用者序列 $US^T_k = < s_1\ s_2 \cdots s_n >$，若 $Y = < s_{i1}\ s_{i2} \cdots s_{im} >$為 X 在 US^T_k 所謂對應的使用者序列 US 中的一個最小發生，則 MO 在 US^T_k 中的效益如式(3)所示。

$$su(MO) = \sum_{i_j \in MO} su(i_j, MO) = \sum_{i_j \in MO} su(i_j, US^T{}_k) \tag{3}$$

若 PB = {MO_1, MO_2, …, MO_q}為 X 在 US^T_k 中的一個購買行為，則 PB 的效益如式(4)所示。

$$su(PB) = \sum_{MO_i \in PB} su(MO_i) \tag{4}$$

而序列 X 在使用者交易序列 US^T_k 中的效益如式(5)所示。

$$su(X, US^T_k) = \max\left(su(PB_i) \middle| PB_i \in PB(X, US^T_k)\right) \tag{5}$$

其中 PB(X, US^T_k)表示 X 在 US^T_k 中可找出的所有 PB 的集合。而序列 X 在整個序列資料庫 SDB 中的效益如式(6)所示。

$$su(X, SDB) = \sum_{US^T_i \in SDB} su(X, US^T_i) \tag{6}$$

藉由使用者自訂之最小效益門檻率（Minimum Utility Threshold）δ，我們可求得最小效益門檻值（Minimum Utility）MU =δ × su(SDB)；若一序列 S 在序列資料庫 SDB 中的效益 su(S, SDB) ≥ MU，則我們稱 X 為高效益序列型樣（High Utility Sequential Pattern），簡稱為 HUSP。

與之前的研究[7]相比，我們的定義除了界定清楚何謂一個序列在使用者序列中的發生，最大的差別在於增加了最小發生的概念。在先前的研究中[7]，計算一個序列 X 在使用者交易序列 US^T 中的效益前，需找出其所有 X 在 US^T 中的發生，接著從所有可能的發生組合中找出彼此互斥且效益總合最高的發生集合。然而我們在找序列型樣時，最主要的目的是找出彼此有因果關係的商品，例如顧客因為買了印表機，之後才來買碳粉匣；而一般情況下，購買時間較近的商品我們認為這樣的因果關係也較顯著，例如顧客先買了 A 廠牌的印表機，之後又買了 B 廠牌的印表機，再之後又買了 B 廠牌的碳粉匣，而買 B 廠牌的碳粉匣自然是因為他最近購買了 B 廠牌的印表機，而非較久之前購買了 A 廠牌的印表機；也因此我們在計算效益前須先找到序列 X 在使用者交易序列 US^T 中的最小發生，如此一來，不但我們不用找所有可能的發生並將其進行所有可能的排列組合，同時又能找到真正具有關聯性的商品序列。此外，我們的方法與先前研究[7]的方法在找尋可能成為高效益序列型樣的序列時都採用某種高估方式來取代序列的實際效益值並藉此提前排除不可能的序列，雖然由於高估其效益的關係使許多不是高效益序列型樣的序列在探勘過程中也要特別對其計算，但我們的方法所採用的高估值不但比先前研究[7]採用的高估值更低，且在探勘過程中往實際效益值收斂的速度更快，讓我們可以在不遺失任何高效益序列型樣的條件下更快排除不必要計算的序列；同時，先前研究的方法[7]在找出所有可能成為高效益序列型樣的序列之後才可以開始計算這些序列的實際效益值，不但需要重新掃描原始資料庫，其計算方式也相當繁複，而我們的方法則可在

探勘過程中在不額外增加掃描成本的條件下找到這些序列的實際效益值，讓整體的執行效率得到大幅度的提升。

壹、相關研究(Related Work)

1995 年由 RakeshAgrawal 等人提出 AprioriAll 演算法[2]來解決從交易資料庫中找出頻繁序列型樣的問題，其方法可分為 5 大步驟：(1) 將原始資料庫根據 customer-id 與購買日期進行排序，並重新整理為序列資料庫的形式。(2) 計算各個項目集的*支持度（support）*，並篩選出那些項目集的支持度超過使用者自定的最小支持度門檻，稱其為*頻繁項目集（frequent itemset）*。(3) 對這些頻繁項目集給予新的編號，再將原始資料庫中的每個使用者序列根據這個編號重新編碼。(4) 將每個重新編碼過的頻繁項目集當作長度為 1 的*序列型樣(sequential pattern)*，彼此兩兩組合產生可能為長度為 2 的*候選序列(candidate sequence)*；之後，再掃描一次資料庫並計算得到各候選序列的支持度，再藉由最小支持度來篩選出長度為 2 的序列型樣；接下來重複相同的步驟，由長度為 k 的序列型樣中找出前(k-1)項相同的序列作組合，產生長度為(k+1)的候選序列，並檢查這些候選序列所有長度為 k 的子序列（subsequence）是否為序列型樣；若有任何子序列並非序列型樣，則將此長度為(k+1)的候選序列刪除；之後，掃描一次資料庫得到所有長度為(k+1)候選序列的支持度並刪除不足最小支持度之序列，得到長度為(k+1)的序列型樣，再依此產生下一個長度的候選序列，直到無法產生出更長的候選序列為止。(5) 將所有找到的序列型樣藉由步驟(3)的編碼對照表還原成原本的項目集，並由長度較長的序列型樣開始，將其的子序列從序列型樣中刪除，只留下*最大序列型樣（maximal sequential pattern）*。此方法透過對頻繁項目集重新編碼來簡化問題，使之後的步驟不需再考慮商品一起購買的情況；並藉由「若一序列的支持度較最小支持度低，則其超序列（supersequence）的支持度亦必定較最小支持度低」的*向下封閉（downward closure）*特性，排除許多不可能是頻繁的序列，留下可能的候選序列後才進行實際計算支持度與篩選。但是這樣的方法，最大的缺點是每產生長度多 1 的候選項目集都要掃描一次原始資料庫並搜尋大量的候選序列以計算其支持度，嚴重影響執行之效率，故整體來說速度與記憶體使用的效率均不佳。

2001 年由 Jian Pei 等人提出了 PrefixSpan 演算法[4]，同樣是解決找出頻繁序列型樣的問題，而其所採取的策略為 pattern-growth 的方式：先掃描一次原始資料庫，找出所有頻繁項目，並以這些頻繁項目作為長度為 1 的序列型樣；接著重新掃描一次資料庫並以各頻繁項目為*前綴（prefix）*產生對應各前綴的*投影資料庫（projected database）*；接著對各投影資料庫重複相同的步驟，掃描一次投影資料庫找出所有的頻繁項目，再將這些頻繁項目與原本長度 k 的前綴組合產生長度為(k+1)的序列型樣，並以這些長度為(k+1)的序列型樣為新的前綴來產生新的投影資料庫，直到找不到任何頻繁項目為止。此方法利用前綴與投影資料庫來找出所有以該前綴開頭的序列型樣，不需產生與搜尋候選序列，且整個過程僅需掃描 2 次原始資料庫，使執行效率得到大幅度的提升；但是在執

行過程中會產生大量的投影資料庫，雖然每次產生的投影資料庫必定會比前一次的資料庫更小，但須同時產生多個投影資料庫，且這些資料庫會持續占用記憶體空間直到被計算並產生下一層的投影資料庫才會被釋放，故記憶體使用量的需求較高。

2005 年由 Y. Liu 等人提出了 Two-Phase 演算法[5]，不同於過去對探勘頻繁項目集與序列型樣之研究，捨棄原本以出現頻率，也就是支持度，作為評估一個項目集或序列是否重要的指標，改使用效益為新的評估指標。因為可能有些商品雖然賣出的頻率較低，但售出時可以帶來更高的利潤，這樣的商品對商家來說重要性可能比那些常常賣出但總獲得利潤不高的商品更為重要；但傳統探勘頻繁項目集的研究中因為只有考慮商品的出售頻率而不考慮其獲利情況，可能使那些真正可以為商家帶來較高利潤的商品無法被找到。然而，一個項目集或序列的效益並不存在「若一個項目集或序列的效益超過最小效益門檻值，則其所有子集或子序列的效益也都會超過最小效益門檻值」這樣的向下封閉性；因此，本篇論文的作者提出了*交易權重效益（transaction-weighted utility*，簡稱為 *twu*）做為項目集效益的估計值，以所有包含此項目集的交易的整體效益值之總和來當作此項目集的效益估計值，並藉此保持向下封閉的特性，即「若一個項目集的 twu 超過最小效益門檻值，則其所有子集的 twu 也都會超過最小效益門檻值」，且因為一個高效益項目集的 twu 也必定會超過最小效益門檻值，先找出符合最小效益門檻值的項目集，必定可以從中找出所有的高效益項目集，因此 Two-Phase 演算法可以分成兩個階段：第一個階段為找出所有 *htwu 項目集*，也就是找出所有 twu 較最小效益門檻值高的項目集；第二階段則是重新掃描原始資料庫，計算每個 htwu 項目集的實際效益，找出所有的高效益項目集。此方法的好處是在第一階段可直接利用找頻繁項目集的方法[1]來找出所有的 htwu 項目集，但第二階段必須掃描原始資料庫，搜尋並計算大量 htwu 項目集的效益；且由於項目集的 twu 常常遠大於其實際的效益，因此很容易產生出許多雖然 twu 超過最小效益門檻值但實際效益卻不足的項目集，使得接下來的第二階段進行許多不必要的搜尋和計算。

之後於 2010 年，由 ChowdhuryFarhan Ahmed 等人所提出的 HUSP 演算法[7]，則是基於上述 Two-Phase 所提出的概念，發展出探勘高效益序列型樣的方法；此方法以*序列權重效益（sequence-weighted utility*，簡稱為 *swu*）來取代原本的交易權重效益 twu，並使用同樣 Two-Phase 的架構來進行探勘：第一階段為找出所有的 *hswu 序列*，即所有 swu 超過最小效益門檻值的序列；第二階段則重新掃描一次資料庫，搜尋並計算所有 hswu 序列的實際效益值以找出所有的高效益序列型樣。本篇論文對於第一階段提出了兩種版本的演算法－UL 及 US，分別是基於 AprioriAll[2]與 PrefixSpan[4]的方法進行修改，藉以找出所有的 hswu 序列型樣。此方法的優缺大致上與上述三篇研究各自的優缺大同小異，不過在第二階段的計算上卻相當複雜；根據此論文所給定的定義，一個序列 X 在一個使用者序列的效益值為，X 在該使用者序列中可找出的所有可能排列組合中效益最高者，例如在使用者序列< A(2) A(1) B(3) C(1) A(4) B(2) C(3) >中求出序列< A B C >的效益，需先找出所有可能的< A B C>的組合，包括{< A(2) B(3) C(1) >}、{< A(2) B(3) C(3) >}、{< A(2) B(2) C(3) >}、{< A(1) B(3) C(1) >}、{< A(1) B(3) C(3) >}、{< A(1) B(2) C(3) >}、{< A(4) B(2) C(3) >}等 7 種序列集，及這 7 種序列可能的兩兩組合、三三組合等，

包括{< A(2) B(3) C(1) >, < A(1) B(2) C(3) >}、{< A(2) B(3) C(1) >, < A(4) B(2) C(3) >}、{< A(1) B(3) C(1) >, < A(4) B(2) C(3) >}、{< A(2) B(2) C(3) >, < A(1) B(3) C(1) >}等 4 種序列集，再從這 11 種序列集中算出效益最高的集合，以此作為此使用者序列提供給 X 的效益；從這個例子中可以發現，這樣的計算不但在進行排列組合時將面臨相當大的難題，且這些組合中又有許多是重複的計算，例如{< A(2) B(3) C(1) >, < A(1) B(2) C(3) >}與{< A(2) B(2) C(3) >, < A(1) B(3) C(1) >}其實組成的項目是一樣的。除了這些計算層面的問題外尚有另一個問題，便是藉由此方法所找出的序列型樣不見得是真正具有意義的，例如上述的例子，最後找到效益最高的序列組合為{< A(2) B(3) C(1) >, < A(1) B(2) C(3) >}或{< A(2) B(2) C(3) >, < A(1) B(3) C(1) >}，其中對於第一種組合來說，< A(2) B(3) C(1) >我們可以發現 A(2)與之後的 B(3) C(1)之間其實還有一個 A(1)，若此顧客是買了 A 之後才會去買 B 與 C，那應該是因為買了 A(1)才會去買之後的 B(3) C(1)，而不是因為買了 A(2)，因為 A(1)購買的時間與 B(3) C(1)購買的時間較近。因此在我們的定義中，我們提出最小發生的概念來避免找出不具有直接關聯性的商品購買順序，以確保所找出的序列型樣都是具有意義的。

貳、我們的方法(Our Algorithm)

本章將介紹我們的方法，以及與我們方法相關的定義；之後將以實際範例敘述我們方法執行之流程，並說明我們方法的特色。

（一） 方法架構

我們的方法以 PrefixSpan 演算法[4]為基礎，一開始對原始序列資料庫進行一次掃描，計算所有項目的序列權重效益 swu 與*投影權重效益（projected-weighted utility，簡稱為 pwu）*，並以使用者自訂的最小效益門檻值 MU 進行篩選，找出所有 swu 大於等於 MU 的項目，又稱為 *hswu 序列*，以及 pwu 大於等於 MU 的項目，稱為 *hpwu 序列*；接著以這些 hpwu 序列作為前綴，並再掃描一次原始序列資料庫，刪去 swu 較 MU 小的項目並產生出對應於各個前綴的投影資料庫，對於前綴為 *Pre* 的投影資料庫我們將其表記為 *Pre*-projected database。接著，對於各個投影資料庫進行重複的相同步驟：(1) 掃描一次投影資料庫，同時計算所有項目的 swu、pwu 並找出此投影資料庫之前綴的購買行為(Purchasing Behavior)來計算其實際效益值。(2) 篩選出 pwu 大於等於 MU 的項目，使其與目前的前綴組合成新的前綴。(3) 再掃描一次投影資料庫，刪去所有 swu 較 MU 小的項目並產生出對應各前綴的下一層投影資料庫。藉由重複此三步驟，直到無法找到任何 hpwu 的項目時便可終止，同時也找出了所有的高效益項目集。

（二） 相關定義

由於序列的效益並沒有向下封閉性，因此，過去利用此特性來快速排除不可能是高效益序列的方法無法使用，所以之前的研究[7]提出了一個具有向下封閉性的估計值，也就是序列權重效益 swu。swu 的原理是將包含某個序列 S 的所有使用者序列的效益值總和，如式(7)所示。

$$swu(S) = \sum_{US_i \supseteq S} su(US_i) \tag{7}$$

從這裡可很明顯發現，對任一包含序列 S 的使用者序列 US_k，由於 $su(S, US_k) \leq su(US_k)$，故 $su(S) \leq swu(S)$；同時，若序列 S 為高效益序列型樣，即 $su(S) \geq MU$，則代表 $swu(S) \geq su(S) \geq MU$，也就是若一序列 S 為高效益序列型樣，則其 swu 大於等於 MU；我們將 $swu(S) \geq MU$ 的序列 S 稱為 hswu 序列，而只有當一序列為 hswu 序列時，此序列才有可能是高效益序列型樣。再者，對於一序列 S 與任一 S 的超序列 S'，可知所有包含 S 之使用者序列的集合，必定是所有包含 S'之使用者序列的集合之超集合，故可知 $swu(S) \geq swu(S')$；若 S'為 hswu 序列，即 $swu(S') \geq MU$，則 $swu(S) \geq swu(S') \geq MU$，可知 S 也必定是 hswu 序列，而這樣的一個性質，就是所謂的向下封閉性。藉由向下封閉性，我們可知道若一序列 S 不是 hswu 序列時，其超序列 S'亦不可能是 hswu 序列，同時也不可能是高效益序列型樣。

由於 swu 是使用所有包含一序列 S 的使用者序列的效益總和當作 S 的估計效益，很容易找出出現頻繁但自身效益不高的序列，使探勘過程浪費不必要的時間；為了使我們的方法比起先前的研究[7]更有效率，我們引入了一些新的概念：使用者序列的*投影效益（projected utility*，簡稱為 *pu*）表示的是一個使用者序列在投影資料庫中的序列效益值，若一使用者 $US_k = < s_1\ s_2 \ldots s_n >$序列在一個以 *Pre* 為前綴的投影資料庫 PDB 中產生的序列為 $PUS_k = < s_i'\ s_{i+1} \ldots s_n >$（$i \geq 1$ 且 $s_i' \subseteq s_i$），則 US_k 於 PDB 中的 pu 如式(8)所示。

$$pu(US_k, PDB) = su(PUS_k) = \sum_{i_j \in PUS_k} su(i_j, PUS^T{}_k) \tag{8}$$

也就是將被排除於投影資料庫外的項目的效益也從投影資料庫中該序列的整體效益中排除；例如原始的序列資料庫中一使用者序列為< A(2) B(3) A(1) C(2) D(1) >，此序列在以< A >為前綴的投影資料庫中對應到的序列為< B(3) A(1) C(2) D(1) >，那麼此序列於< A>的投影資料庫中的投影效益 pu 即等同< B(3) A(1) C(2) D(1) >的序列效益 su。使用者序列的*後綴效益（suffix utility*，簡稱 *sfu*）則是對於一個使用者序列 US_k 中的一個項目 i_p，只計算該項目及位置(location)在該項目之後的所有項目所構成之子序列的序列效益值，如式(9)所示。

$$sfu(i_p, US^T{}_k) = \sum_{i_j \in US_k \wedge loc(i_p, US^T{}_k) \leq loc(i_j, US^T{}_k)} su(i_j, US^T{}_k) \tag{9}$$

最後是一個項目的*投影權重效益（projected-weighted utility*，簡稱為 *pwu*），其計算方式為計算每個包含此項目之使用者序列的 sfu 與在該使用者序列中離此項目最近之前綴的序列效益的總和，如式(10)所示。

$$pwu(i_p) = \sum_{US^T{}_j \supseteq i_p} \left(su(\mathrm{Pre}_j) + sfu(i_p, US^T{}_j) \right) \quad (10)$$

例如上例中 C 在< A >的投影資料庫中的< B(3) A(1) C(2) D(1) >這個序列中的後綴效益 sfu 即為< C(2) D(1) >的序列效益 su，而其投影權重效益 pwu 在此序列中將增加< A(1) >與< C(2) D(1) >的序列效益 su 的總和；在這裡雖然對於< B(3) A(1) C(2) D(1) >來說前綴是< A(2) >，但由於計算 C 的 pwu 等於是在計算以< A C >為前綴的超序列(含< A C >)的序列效益最大的可能值，並考慮到前面所定義的最小發生可發現< A(2) C(2) >並不是一個最小發生而< A(1) C(2) >是，故這裡在計算 C 的 pwu 時將以< A(1) >來取代< A(2) >作為前綴的效益。

若一個項目的 pwu 大於等於 MU 時，我們稱由目前的前綴與此項目所構成的序列為 *hpwu 序列*。由於比起 swu 是將整個使用者序列的效益拿來估計其子序列效益的最大值，pwu 將使用者序列中一些與欲求效益的子序列無關的項目排除在外，但又必定包含該子序列的效益，故可由 $su(US_i) \geq (su(Pre_i) + sfu(i_p, US^T{}_i))$得到 $swu(S) \geq pwu(S) \geq su(S)$這樣的關係；所以若一序列 S 為高效益序列型樣，則 S 必為一個 hpwu 序列。同時，pwu 與 swu 相似，亦保有向下封閉性，也就是若一個序列 S 為 hpwu 序列，則其所有子序列也都會是 hpwu 序列。而我們的方法之所以同時使用 swu 與 pwu 兩種效益估計值，則是分別取其特性，將 swu 不足 MU 的項目直接在投影資料庫中刪除，因為由目前的前綴與該項目組成之序列的超序列必定不是 hswu 序列，亦不可能是高效益序列型樣；而 pwu 較 MU 小的項目，則代表由目前的前綴與該項目所組成之序列為首，之後再增加其他項目的序列必定不是 hpwu 序列，也不可能是高效益序列型樣，故不須產生以這樣序列為前綴的投影資料庫。

（三） 方法執行範例

接下來以表 1 的利潤表與與表 2 的資料庫為例來說明我們演算法的流程。為了之後說明方便，這裡我們先將表 1 中各項目的利潤帶入表 2 的資料庫，可得到表 3；其中，每個項目後面括號裡的數字表示的不是該項目在該次交易中被購買的數量，而是實際帶來的獲利值，即該次交易中該項目被購買的數量乘以單位利潤。

表 3 轉換後的原始資料庫

SID	User-Sequence
10	< a(15) {a(10) b(42) d(20)} f(8) a(25) d(10) >
20	< e(18) {a(10) b(35)} d(10) c(12) >
30	< {c(3) f(16)} b(21) {d(10) e(24)} >
40	< a(10) {b(49) d(40)} {a(30) b(21)} e(30) >
50	< {d(10) f(24)} c(15) g(18) >
60	< d(20) e(6) {a(35) b(56)} d(30) b(42) e(18) >

一開始我們先掃描一次資料庫，首先讀到 SID 為 10 的使用者序列；由於初次掃描時尚未得到整個使用者序列的效益及整個資料庫的效益，故我們在進行對每個項目

的 swu 與 pwu 的計算時先記錄該使用者序列包含哪些項目，且各項目得到的 sfu 與 swu 的差值是多少。掃描第一個使用者序列時，可得到項目 a 的 pwu 要增加 su(US10)、項目 b 的 pwu 要增加 su(US10) – 25、項目 d 的 pwu 要增加 su(US10) – 67、項目 f 的 pwu 要增加 su(US10) – 87，而當第一個使用者序列掃描完畢時，可得到整個使用者序列的效益值為 130，便可更新上述 4 個項目的 pwu 值，且這 4 個項目的 swu 均增加 130。接下來一邊掃描剩的使用者序列一邊做相同的計算，可得到各項目的 swu 與 pwu 如表 4 所示，而整個序列資料庫的效益為 743；假定使用者自訂之最小效益門檻率 δ=30%，則可得知最小效益門檻值 MU = 743 × 30% = 223。

表 4 原始資料庫中各項目的 swu 與 pwu

Item	a	b	c	d	e	f	g
swu	602	676	226	743	546	271	67
pwu	558	533	119	514	326	171	18

接下來對每個項目的 swu 與 pwu 與 MU 比較，可發現 swu(g) = 67 < MU，所以之後產生投影資料庫時可將 g 直接排除，因為任何包含 g 的超序列都將不可能是 hswu 序列；又因為 pwu(c) = 119 < MU、pwu(f) = 171 < MU，代表 c 或 f 開頭的序列不可能是 hpwu 序列，所以之後不用以 c 或 f 作為前綴產生投影資料庫，而要以 a、b、d、e 作為前綴產生其對應的投影資料庫。在對 a 產生投影資料庫時，我們依序找出所有包含 a 的使用者序列，並從其中找出第一個 a 出現的位置，將之後的子序列作為投影資料庫中的使用者序列；之後再將被移除的部分的效益從 pu 中減去，並記錄當下前綴的效益，可得到表 5 的結果。以相同的方法產生< b >、< d >、< e >的投影資料庫，接下來便對各投影資料庫進行下一階段的運算。

表 5 < a >的投影資料庫

SID	**User-Sequence**	**pu**	**Prefix**
10	< {a(10) b(42) d(20)} f(8) a(25) d(10) >	130 – 15= 115	< a(15) >
20	< {_ b(35)} d(10) c(12) >	85 – 28= 57	< a(10) >
40	< {b(49) d(40)} {a(30) b(21)} e(30) >	180 – 10= 170	< a(10) >
60	< {_ b(56)} d(30) b(42) e(18) >	207 – 61= 146	< a(35) >

首先先對< a >的投影資料庫進行運算。一開始同樣是掃描一次投影資料庫，對每個項目計算其 swu 與 pwu，但同時也要找出在每個使用者序列中所有< a >可能的購買行為 PB 並求出效益最高者，藉此得到< a >的序列效益。當掃描到 SID 為 10 的使用者序列時，先掃描到 a(10)這個項目，由於 a(10)也可以獨立構成< a >的一個最小發生，故< a >的序列效益需增加 10，而 a 的 swu 增加 SID 為 10 的使用者序列的 pu 值與前綴 a(15)的效益值，也就是增加 115 + 15 = 130，而 a 的 pwu 則是增加縣在 a(10)的 sfu 與前綴 a(15)的效益值，同樣也是 115 + 15 = 130；接下來掃描到 b(42)時，使 b 的 swu

增加 130、pwu 增加 105 + 15 = 120；同時，因為 b(42)與 a(10)一起購買，也要考慮到{a b}的情況，所以同時也要對_b，表示 b 與目前前綴的最後一個項目集是一起購買的情況，的 swu 增加 130、pwu 則增加 105 + 10 = 115。掃瞄到 d(20)時，使 d 的 swu 增加 130、pwu 增加 63 + 15 = 78；且與 b 的情況相同，也要考慮到{a d}的情況，故_d 的 swu 增加 130、pwu 增加 63 + 10 = 73。當掃描完 d(20)時，由於對之後的序列而言在原本的前綴 a(15)的後面又出現了新的前綴 a(10)，根據最小發生的定義，之後出現項目需要以 a(10)為其前綴組成更長的序列，故接下來掃描到的項目在 pwu 的計算上均需將原本的 a(15)取代為現在的 a(10)；而 swu 的計算必須考慮到出現在目前項目之前的項目，故不需將前綴更新為 a(10)；所以接下來掃描到 f(8)時，其 swu 仍是增加 130、但 pwu 則是增加 43 + 10 = 53。之後掃描到 a(25)與 d(10)由於之前掃描到 a(10)與 d(20)時已將 a(25)與 d(10)的效益計算進去，故不需對 a 與 d 的 swu 或 pwu 再更新，但 a(25)須更新< a >的序列效益，使其增加 25 為 35。而當整個使用者序列被掃描完畢後，還需要將< a >的序列效益再加上原本這個序列的前綴< a(15) >的效益，得到 SID 為 10 的使用者序列對< a >提供了 50 的效益，且這是< a >在這個使用者序列中唯一的 PB。之後以同樣方式掃描剩餘的使用者序列並計算各項目的 swu 與 pwu，以及前綴< a >的序列效益，得到表 6 的結果與< a >的序列效益為 135，此時可知< a >不是高效益序列型樣。

表 6 < a >的投影資料庫中各項目的 swu 與 pwu

Item	a	b	c	d	e	f	_b	_d
swu	310	491	67	558	361	130	558	130
pwu	221	395	22	366	113	53	444	73

接下來以 MU 對每個項目的 swu 與 pwu 進行篩選，發現 swu 較 MU 小的項目為 c、f、_d，代表這三個項目在下一層的投影資料庫產生時均可排除；而 pwu 大於等於 MU 的項目為 b、d、_b，與前綴分別結合成< a b>、< a d >與< {a b} >，並以這三個序列作為新的前綴產生對應的投影資料庫，並重複相同之步驟，即可找出所有以< a >為開頭的高效益序列型樣。之後，再到< b >、< d >、< e >的投影資料庫進行相同的運算，即可找出所有的高效益序列型樣，結果如表 7。其中可發現，我們的方法在運算過程中一共找出 23 個 hpwu 序列，也就是產生了 23 個不同的投影資料庫；與之前僅使用 swu 作為估計值的方法[7]找出 45 個 hswu 序列相比，我們不但省去許多不必要的計算，同時也省去找出所有 hswu 序列後還要重回原始資料庫計算各序列實際效益的動作，得到速率上的大幅改進。

表 7 範例的輸出結果

Prefix	High Utility Sequential Pattern	# of hpwu sequence
< a >	< a {b d} a > : 231, < a d b e > : 226, < {a b} > : 239, < {a b} d > : 238	9

< b >	< b > : 266	4
< d >	< d {a b} e > : 250	6
< e >	None.	4

（四） 方法的特色

比起過去的方法[7]，我們的方法主要改進的地方在大幅降低序列效益的高估情況，且隨著不斷產生投影資料庫，pwu 也會越來越接近實際的效益，讓不可能成為高效益序列型樣的序列提早被排除，同時減少運算量及避免產生不必要的投影資料庫；除此之外，與原本的 Two-Phase Based 方法相比，我們的方法不需要在最後重新掃描一次投影資料庫來重新計算序列的實際效益，直接在投影資料庫中一面計算各個項目的 swu 與 pwu 時一面算出 hpwu 序列的實際效益，同時也可以簡單找出最小發生，省去列出所有排列組合並計算的麻煩。

參、結論(Conclusion)

本篇論文對序列效益提出新的定義，強調項目與項目之間在被購買時的因果關係，以解決過去由於定義欠佳導致計算複雜且找出的序列型樣未必具有應用價值的問題；除此之外，本篇論文亦提出一個新的高效益序列型樣探勘演算法，不但能有效且快速的鎖定可能是高效益序列型樣的序列，同時又可以避免找出許多不必要的候選序列，並利用投影資料庫的方式將需要檢查的資料縮小到只剩與這些可能是高效益序列型樣的序列有關的部分，一面找出更長的候選序列一面計算候選序列的實際效益，將原本繁複的工作一次解決又不額外增加工作量。未來本研究將繼續使目前的方法更加有效率，並實際以實驗比較我們的方法與過去研究的方法執行時的效能差；此外，我們也希望可嘗試捨去最小效益門檻，將問題改為尋找前 k 個效益最高的序列型樣，以供使用者可以多一種選擇來制定決策。

參考文獻(Reference)

1. R. Agrawal and R. Srikant, “Fast Algorithm for Mining Association Rules,” *International Conference on Very Large Data Bases*, 1994, pp.487-499.
2. R. Agrawal and R. Srikant, “Mining Sequential Patterns,” *International Conference on Data Engineering*, 1995, pp.3-14.
3. R. Agrawal and R. Srikant, “Mining Sequential Patterns: Generalizations and Performance Improvements,” *International Conference on Extending Database Technology*, 1996, pp.3-17.
4. J. Pei, J Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Daya and M.C. Hsu, “PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth,” *International Conference on Data Engineering*, 2001, pp.215-224.
5. Y. Liu, W. Liao, and A. Choudhary, “A Fast High UtilityItemsets Mining Algorithm”, *Proc. Of the ACM Intel. Conference on Utility-Based Data Mining Workshop (UBDM)*, 2005.

6. S.J. Yen, C.C. Chen, and Y.S. Lee, “A Fast Algorithm for Mining High Utility Itemsets,” *PAKDD 2011, LNAI*, 2011, pp.171-182.
7. C.F. Ahmed, S.K. Tanbeer, and B.S. Jeong, “A Novel Approach for Mining High-Utility Sequential Patterns in Sequence Databases,” *ETRI Journal* (32:5), 2010, pp.676-686.
8. J.W. Huang, S.C. Lin, and M.S. Chen, “DPSP: Distributed Progressive Sequential Pattern Mining on the Cloud,” *PAKDD 2010, LNAI*, 2010, pp.27-34.
9. MOHAMMED J. ZAKI, “SPADE: An Efficient Algorithm for Mining Frequent Sequences,” *Machine Learning*, 2001, pp.31-60.