新的 pCluster方法: pCluster+ 與 incremental pCluster

李建億

台南師範學院資訊教育研究所

E-mail:leeci@ipx.ntntc.edu.tw

黃乙展

台南師範學院資訊教育研究所

E-mail:ichain@ms45.hinet.net

吳崢榕

台南師範學院資訊教育研究所

E-mail:wjr@benz.ntntc.edu.tw

摘要

分群是一種將相似的物件或是屬性分類到同一群的過程，傳統的分群技術大都採用計算物件與物件之間的"相似度"來進行分群，而相似度的計算多以物件間的距離為基礎，例如:歐幾里得距離等等。 Haixun Wang **等人在**[5]**提出了一個新的分群模型**"pCluster"**，有別以往的分群法，兩個物件之間是否相似，取決於它們維度中的子集是否有相同起伏的區塊。如何在大量資料中準確、有效率的找出這樣的群，便成了一個很有趣的問題。在本篇論文中，提出一個新的演算法**:pCluster+**，使得處理** pCluster **的問題更為快速，同時也提出解決資料量新增**(Incremental)**問題的方法。經過我們實驗証明，本篇論文所提出的方法都要比原來的** pCluster **方法快上好幾倍。**

關鍵詞：分群法、相似度、資料新增

一、緒論

分群的技術可分為多種方法，並被廣泛的應用在各種領域上，像是統計、機器學習、迴歸分析、影像處理、及基因分析等等，其中有許多的研究在探討如何在一個大的資料集中，有效率的進行分群分析，大部分研究的貢獻在於分群方法的延展性及處理高維度分群的技術。在高維度分群時，由於資料複雜度高，因此分群的結果常帶有些不確定性[8]**，近來也有些研究**[1,2,3,4,9]**探討埋藏在高維度子空間裡的群。而這些分群法大都是以距離來做為相似度的計算基礎，但在某些情況下以距離來計算相似度並不適當，像是基因對實驗反應的分群，必須在高維度的資料中找出有意義的區塊。舉個例子來說，圖** 1-1 **是一個** 5x5 **的資料集，其中** *O* **代表著物件編號，而** *C* **則是維度。表格中的值則代表某種實驗或是反應所呈現的值，例如:基因(物件** *O*)**在不同實驗(維度** *C*)**中所產生的反應。如果以傳統距離為基礎的分群法進行分群，則可能無法得出任何有意義的結果。將圖** 1-1 **轉換成折線圖，如圖** 1-2 **所示，同樣的圖** 1-2 **中很難用肉眼或傳統的分群法找出隱藏在其中的關係，如果只看其中的** $\{(O_1,O_2,O_4)=>(C_0,C_3,C_4)\}$ **並轉換成折線圖** 1-3**，則可以發現三條具有相同起伏且互平行的曲線。這三條曲線如果以距離來計算相似度可能不具有任何關係，但從圖中卻可以發現有很強烈的關連存在。我們可以說** O_1、O_2、O_4 **這三個基因，在** C_0、C_3、C_4 **這三種情況下會有相似的行為表現。而這樣的關係就稱為一個** pCluster(Pattern Cluster)**。**

	C_0	C_1	C_2	C_3	C_4
O_0	995	942	827	436	391
O_1	359	604	902	963	465
O_2	379	153	292	983	485
O_3	382	421	716	718	895
O_4	259	447	726	863	365

圖 1-1 5x5 **資料集**

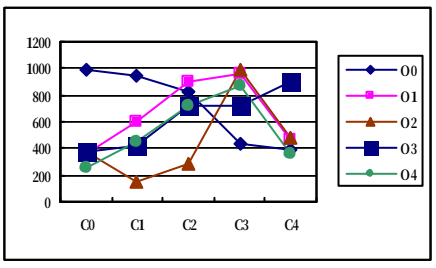

圖 1-2 **將資料集以折線圖表示**

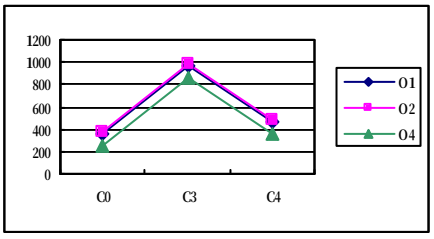

圖 1-3 **以折線圖表示** 3x3 pCluster

如同上面的例子，pCluster 模型可以被應用在許多的方面，像是基因工程中，找出具有相同行為反應的基因。在電子交易中，分析出具相同喜好的顧客等等。pCluster 可以算是一種高維度子空間分群的問題，其定義又比高維度子空間的分群廣泛些，以區塊間的相似度取代原有的距離為礎的測量。解決 pCluster 這類的問題的方法主要是由 Haixun Wang 等人在[5]中所提出，Haixun Wang 等人所提出的 pCluster 演算法在處理大型資料時仍需要不少的時間，本篇論文將提出一個更快速的演算法來解決這個問題。

另外，資料新增的問題也十分值得去探討及研究。所謂"incremental"是指，當新的物件不斷的增加到原本的資料中，如何才能比重做整個演算法更有效率，在本篇論文中，也將針對這個問題，提出解決的方法。本論文的其它章節為:第二章，介紹一些相關的研究，以及對 pCluster 演算法大略的介紹，第三章則是本篇論文所提出新的演算法:pCluster+，第四章則是探討新增資料的問題。第五章將對這些方法進行效能的分析與測試，第六章為本論文之結論。

二、相關研究

在高維度空間的分群法，已有許多人提出，但多是以距離來做為分群的基礎，因此如果將這些方法用來處理 pCluster 的問題就顯得不恰當，而且找出來的答案也會有問題。Cheng 等人在[11]中提出了"Bicluster"的觀念它可說是 pCluster 的前身。Bicluster 主要是在一個 DNA 的陣列中找出連貫的基因及情況，而它使用了平均平方差的計算方式來做為分群的基礎。公式如下:

$$H(I,J)=\frac{1}{|I||J|}\sum_{i\in I,j\in J}(d_{ij}-d_{iJ}-d_{Ij}+d_{IJ})^2$$

$$d_{iJ}=\frac{1}{|J|}\sum_{j\in J}d_{ij},\quad d_{Ij}=\frac{1}{|I|}\sum_{i\in I}d_{ij},\quad d_{IJ}=\frac{1}{|I||J|}\sum_{i\in I,j\in J}d_{ij}$$

假設 *X*、*Y* 分別代表著基因及各種情況，*I*、*J* 分別屬於 *X*、*Y*，如果 H(*I*,*J*)≦δ，δ為一門檻值，則稱 *Aij* 為一個 δ-bicluster。

Yang 等人在[6]提出了一個以移動為基礎的演算法，它能夠更快且更有效率的找出 Biclusters。從一個隨機選擇的起點開始，不斷的重覆執行以改善群的品質。避免了原先群跟群重？的問題，也能同時找出多個群，但仍是以平均平方差來做為分群的基礎，所以當遇到資料中存在雜質的情況，就使得分群的結果不盡理想，同時需要分群數目來做為輸入參數。

而 Haixun Wang 等人在[5]所提出的 pCluster 概念，以 pScore 取代了平均平方差算方式，很有效的解決了之前兩個方法所遇到的問題，不論是雜質的問題或是重？的問題，同時也聲稱能夠找出所有隱藏在資料集的 pClusters。pScore 公式如下:

$$pScore\left(\begin{bmatrix} d_{xa} & d_{xb} \\ d_{ya} & d_{yb} \end{bmatrix}\right)=|(d_{xa}-d_{xb})-(d_{ya}-d_{yb})|$$

d_{mn} 為物件 m 在屬性 n 的值，如果(*O*,*T*)是一個 δ-pCluster，其中 *O*、*T* 分別為物件及屬性的子集合，則(*O*,*T*)中任何的 2x2 的子陣列 *X* 都要符合 pScore(*X*) ≦ δ 的條件。

接下來分別對所有的物件及屬性進行兩兩的比對，產生 MDS(Maximum Dimension Sets)，物件所產生的 MDS 稱為 MDSo，而屬性則稱 MDSc。所產生的 MDSo 及 MDSc 包含了整個資集中所有可能成為 pCluster 的資訊，但也包含了許多不必要、多餘的資訊，因此必需對 MDS 進行砍除的工作。砍除 MDS 的步驟可說是整個 pCluster 的核心，砍除原如下:假設 T_{xy} 表示物件 *x* 跟 *y* 的 MDS，O_{ab} 為維度 *a* 跟 *b* 的 MDS，對任何一個存在 MDS Txy 當中的維度 a 來說，計算 *Oab* 當中包含有{*x*,*y*}的個數，如果小於 *nc*-1(*nc*:最小成為一 pCluster 所需要的維度數目)，則將維度 *a* 從 T_{xy} 中砍除，如果砍除的動作造成了$|T_{xy}|<nc$，則砍除整個 T_{xy}。利用 MDSo 及 MDSc 相互比對砍除，砍掉不必要的部分，一直重覆到沒有任何資訊被砍除為止。將剩餘的 MDS 填入一個名為 prefix tree 的結構中，樹中的每一個結點可以看成一個 pCluster 的候選者，最後再以後序搜尋得出 pCluster 的結果。

三、新的 pCluster 方法: pCluster+

從上一章的介紹中，不難發現一個明顯的問題，當產生 MDS 時必需分別對物件及屬性進行兩兩的比對，如果有一個資料集有上萬筆的物件，那麼比較次數將會有上億次之多，使整個演算法的效能大打折扣。因此本篇論文提出新的 pCluster 方法:pCluster+。

原本的 pCluster 方法需要分別對物件屬性進行兩兩的比較，在一般情況下，物件的數目常會大於屬性許多，因此花在對物件進行兩兩比對來產生 MDSo 的時間將會遠大於對屬性進行兩兩比對產生 MDSc 的時間，舉個例子來說，假設有一個 3000 物件 x30 屬性的資料集，將最小的物件數 *nr* 設為 27，最小屬性數 *nc*

設為 6，而門檻值 *t* 為 1，則產生物件的 MDSo 時間約為 40 秒，產生屬性的 MDSc 則只需 1 秒左右。兩者的差距足足有數十倍之多。因此如果能夠只對屬性產生 MDSc，而不對物件產生 MDSo，將對效能的改善十分有幫助。(如果物件小於屬性則對物件產生 MDSo)。

pCluster+方法的第一步便是對資料集的屬性欄位兩兩比對，產生 MDSc，舉個例子來說，假設有一 6(物件 *O*)x5(屬性 *C*)的資料集如下圖 3-1。其中 *nr*(最小成為一 pCluster 所需要的物件數)等於 3、*nc*(最小成為一 pCluster 所需要的屬性數)等於 3，而門檻值 *t*(等同於上一章所提到的 δ) 為 1。則可以產生如圖 3-2 的 MDSc。

	C_0	C_1	C_2	C_3	C_4
O_0	9	3	6	100	29
O_1	10	5	8	10	56
O_2	7	1	5	6	53
O_3	200	19	22	24	70
O_4	372	-8	-4	-3	43
O_5	11	5	9	11	57

圖 3-1 6x5 資料集

(C_0 C_1) -> (O_0 O_1 O_2 O_5)
(C_0 C_2) -> (O_0 O_1 O_2 O_5)
(C_0 C_3) -> (O_1 O_2 O_5)
(C_0 C_4) -> (O_1 O_2 O_5)
(C_1 C_2) -> (O_0 O_1 O_2 O_3 O_4 O_5)
(C_1 C_3) -> (O_1 O_2 O_3 O_4 O_5)
(C_1 C_4) -> (O_1 O_2 O_3 O_4 O_5)
(C_2 C_3) -> (O_1 O_2 O_3 O_4 O_5)
(C_2 C_4) -> (O_1 O_2 O_3 O_4 O_5)
(C_3 C_4) -> (O_1 O_2 O_3 O_4 O_5)

圖 3-2 MDSc

如同原本的 pCluster 方法，pCluster+方法在產生出 MDSc 後也會有雜質存在，因此必需將雜質去除。在原來的 pCluster 方法中，利用 MDSo 及 MDSc 的相互比對來進行砍除的動作，但 pCluster+方法只有產生 MDSc，因此必需使用新的砍除策略來修剪 MDSc。以下圖 3-3 的形式來表示 MDSc。

MDSc 中的前兩個屬性欄位的值分別命名為 *firstValue* 及 *secondValue*，後面的欄位則是代表物件的值，每一物件欄位都給一個 *object'count* 的初始值 1，另外一個變數是 *MDS_num*，初始值也是 1。

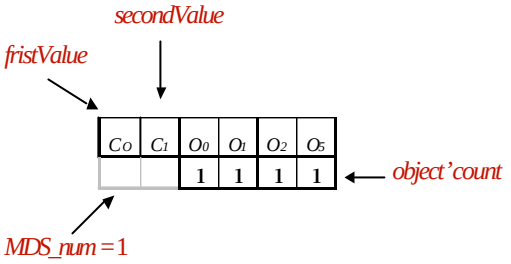

圖 3-3 MDSc 的呈現結構

對所有的 MDSc 進行比對，比對原則如下:

1. 如果它們的 *firstValue* 相等，且 *secondValue* 不相等，則比對 MDSc 的物件欄位。
2. 如果比對後發現，它們物件欄位相同的部分大於等於 *nr*(最小成為一 pCluster 所需要的物件數)，則將相同部分的 *object'count* 加 1，同時合併它們屬性欄位，並將 *MDS_num* 加 1。

以第一個和第二個 MDSc 來說明上面的方法，圖 3-4。因為 C_0 等於 C_0 但 C_1 不等於 C_2，所以必需比對它們的物件欄位。兩個 MDSc 的物件欄位共有 4 個相同，大於所定的 *nr*，3。因此將第一個 MDSc 中相同的物件欄位的 *object'count* 加 1，同時將 *MDS_num* 加 1，並合併兩者的屬性欄位。將所有 MDSc 比對後可以得到如圖 3-5。

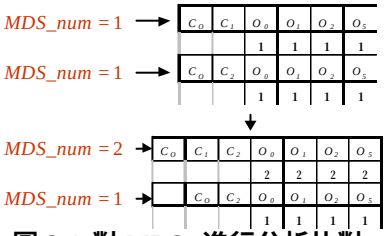

圖 3-4 對 MDSc 進行分析比對

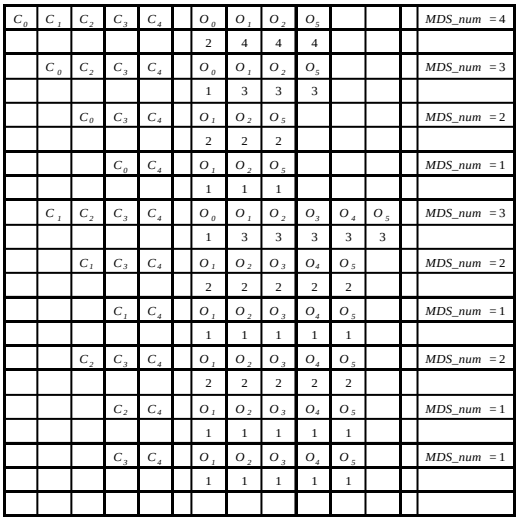

圖 3-5 分析後的果

接下來，對每個 MDSc 中的物件欄位進行分析，以決定是否要砍除。分析砍除的原則如下:

1. 如果其 *MDS_num* 小於 *nc*-1 則將此 MDSc 給砍除。
2. 如果其 *object'count* 小於 *nc*-1 則將此物件砍除。
3. 如果"砍除物件"這個動作造成了物件欄位數目小於 *nr*，則將此 MDSc 砍除。

在砍除完畢後，再將所有存在的子集合併，則成為 pCluster 候選者圖 3-6。

pCluster 候選者

C_0	C_1	C_2	C_3	C_4		O_0	O_1	O_2	O_5			$MDS\_num = 4$
						2	4	4	4			
	C_1	C_2	C_3	C_4		O_1	O_2	O_3	O_4	O_5		$MDS\_num = 3$
						3	3	3	3	3		

圖 3-6 pCluster 候選者

每個候選者其實就是整個資料集的一個子集。接下來對每個候選者而言，如果它們的每個 *object'count* 都等於屬性欄位的數目-1，則可以直接輸出成為一個 pCluster，否則就必須對候選者進行產生 MDSo 的動作。然後將屬性欄位相同的 MDSo 合併，輸出成為 pCluster。

在上面的例子當中，第二個候選者的每個 *object'count* 都剛好等於屬性欄位的數目減 1，所以將第二個候選者直接輸出為一個 pCluster{(O_1,O_2,O_3,O_4,O_5)=>(C_1,C_2,C_3,C_4)}。但第一個候選者則必需要進行產生 MDSo 的動作，產生的 MDSo 如圖 3-7。

(O_0 O_1) -> (C_0 C_1 C_2)
(O_0 O_2) -> (C_0 C_1 C_2)
(O_0 O_5) -> (C_0 C_1 C_2)
(O_1 O_2) -> (C_0 C_1 C_2 C_3 C_4)
(O_1 O_5) -> (C_0 C_1 C_2 C_3 C_4)
(O_2 O_5) -> (C_0 C_1 C_2 C_3 C_4)

圖 3-7 合併相同欄位之 MDS

將相同的屬性欄位 MDSo 合併，則可產生兩個 pCluster{(O_0,O_1,O_2,O_5)=>(C_0,C_1,C_2)}、{(O_1,O_2,O_5)=>(C_0,C_1,C_2,C_3,C_4)}。因此這個資料集共存在三群 pCluster。

雖然上的步驟已經能產生出 pCluster的答案，但在某些情況下，仍會有一些 pCluster 隱藏在剩餘的資料當中，因此必需對剩餘的資料再做處理。重覆上面的砍除動作，產生候選者，然後產生 pCluster，一直到沒有任何新增的 pCluster 為止。總結來說，pCluster+方法，可以用圖 3-8 來表示。

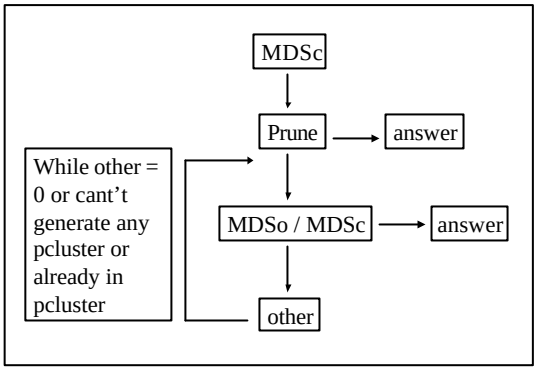

圖 3-8 pCluster+方法流程

pCluster+方法雖然可以很快速的找出所要的答案，但從演算法中也不難發現，pCluster+方法需要額外的記憶體空間來儲存 object'count、MDS_num，並需要額外的時間來進行分析。此外，pCluster+方法在處理剩餘資料上是以迴圈的方式來進行，如同原本 pCluster 中砍除部分的演算法，迴圈執行次數愈多效能表現將愈不理想(此部分純屬理論推測，在我們的實驗中，原本的 pCluster 方法有出現類似的情形，圖 5-5、5-6，但 pCluster+方法則沒有)。

四、Incremental pCluster 的作法:擴展 pCluster+

如同之前所提到的，如果今天有一個新的物件要加入整個資料集，重做整個演算法必然是沒有效率的。在此提出處理資料新增(Incremental)問題的解決方法，而本論文所提出資料新增的 pCluster 方法基本上與上一章的 pCluste+方法很類似，但卻有細部上的不同。這個資料新增的 pCluster 方法可以說是 pCluste+方法的擴展。許多分群的資料新增方法都是針對某種特定的演算法所設計的，但本論文的方法在執行時只需要一個原本的資料集、一個新增的資料集，以及原本資料集的 pCluster 結果便可進行。也就是說不管任何的 pCluster 演算法，都可以用本論文所提出的方法來進行資料新增。

同樣以一個例子來說明資料新增的 pCluster 方法，圖 4-1 是一個 5x5 的資料集，其中 $nr = nc = 3$, $t = 1$。假設新增一個物件 O_5。首先，對 O_5 進行 I_MDS，I_MDS 是以新增的物件為基準，對原來的資料集進行 MDS 的動作，所產生出來的 MDS 稱為 I_MDS。產生的 I_MDS 如圖 4-2。

	C_0	C_1	C_2	C_3	C_4
O_0	9	3	6	100	29
O_1	10	5	8	10	56
O_2	7	1	5	6	53
O_3	200	19	22	24	70
O_4	372	-8	-4	-3	43

O_5	11	5	9	11	57

圖 4-1 5x5 資料集 O_5 為增量物件

(O_0 O_5) -> (C_0 C_1 C_2)
(O_1 O_5) -> (C_0 C_1 C_2 C_3 C_4)
(O_2 O_5) -> (C_0 C_1 C_2 C_3 C_4)
(O_3 O_5) -> (C_1 C_2 C_3 C_4)
(O_4 O_5) -> (C_1 C_2 C_3 C_4)

圖 4-2 I_MDS

在原來的 pCluster 方法中以每個物件(屬性)兩兩比對的方式來產生 MDS，但在此只以 O_5 為基準對原來資料集中的物件進行比對，確保如此不會造成某些資訊遺失的証明如下:

設有 n 個物件，其所產生之 pCluster 之物件集合為 a。其 MDSo 之物件集合為 b，$b >= a$，故 b 包含了 n 個物件中所有可能成為 pCluster 之資訊---(1)。設有 $n+1$ 個物件，其所產生之 pCluster 之物件集合為 c，而其 MDSo 物件集合為 d，由(1)，d 包含了 $n+1$ 個物件中所有可能成為 pCluster 之資訊。設第 $n+1$ 個物件與前 n 個物件所產生之物件集合(I_MDS)為 e。

則 $e >= MDSo(n+1) - MDSo(n)$

$e >= d - b$

$e + b >= d$ 且 $d >= c$

$e + b >= c$

故 $e + b$ 可找出所有的 pCluster。

接下來，以如同 pCluster+方法中所提到的表示法來表示 I_MDS，但進行比對的原則則有所不同，由於是對物件進行 MDS，所以前兩個欄位為物件，後面都是屬性。比對原則如下:

1. 由於 *secondValue* 永遠都是 O_5，因此只要 *firstValue* 不相等，即比較其屬性欄位。
2. 如果比對後發現，它們屬性欄位相同的部分大於等於 nc，則將相同部分的 *column'count* 加 1，同時合併它們的物件欄位，並將 *MDS_num* 加 1。

	O_0	O_1	O_2	O_5		C_0	C_1	C_2				*MDS_num* = 3
						3	3	3				
O_1	O_2	O_3	O_4	O_5		C_0	C_1	C_2	C_3	C_4		*MDS_num* = 5
						2	4	4	4	4		
	O_2	O_3	O_4	O_5		C_0	C_1	C_2	C_3	C_4		*MDS_num* = 3
						1	3	3	3	3		
		O_3	O_4	O_5		C_1	C_2	C_3	C_4			*MDS_num* = 2
						2	2	2	2			
			O_4	O_5		C_1	C_2	C_3	C_4			*MDS_num* = 2
						1	1	1	1			

圖 4-3 I_MDS 分析後的結果

分析結果如圖 4-3。砍除及合併子集的原則，除了將 nr 換置成 nc、nc 換成 nr、*object'count* 變成 *column'count* 之外，其餘皆相同。產生侯選者與對侯選者進行 MDS 則完全相同，最後將侯選者所產生的 pCluster 與原來資料集所產生的 pCluster 進行合併，即是答案，圖 4-4。

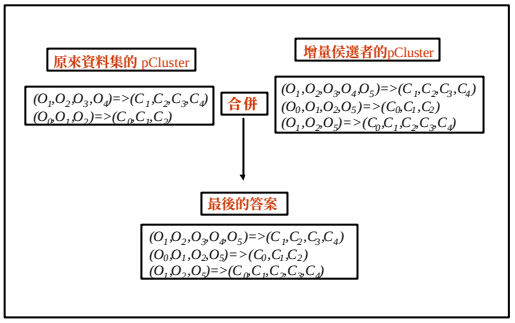

圖 4-4 將兩個結果合併產生答案

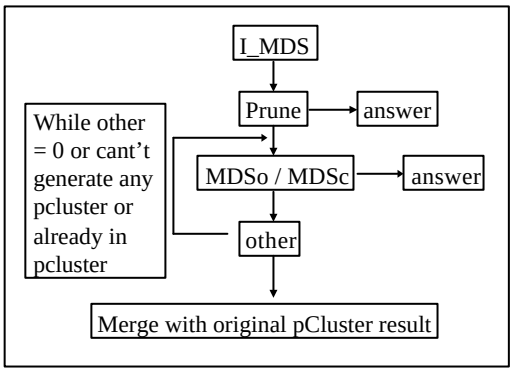

圖 4-5 Incremental pCluster 方法流程

圖 4-5 表示整個資料新增的 pCluster 方法。另外，pCluster 刪除的方法，只要將 pCluster 答案中將想刪除的物件或是維度直接去掉即可，而更新只需先進行刪除，再進行資料新增的 pCluster 方法即可達成。

資料新增的 pCluster 方法為 pCluster+方法的擴展，因此也繼承了大部分的優缺點，如

同 pCluster+方法，它也需要額外的空間來進行儲存，及額外的時間來進行分析，同樣的迴圈執行次數的多寡也將影響效能的表現。

五、效能分析

在這一章中，將對本論文所提出來方法進行效能的分析，實驗平台為處理器 AMD Drun 1.2G，記憶體 384MB 作業系統為 MS WINDOWS 2000，在這一章中，將對不同的資料集，及不同的設定進行實驗。

(一) pCluster+方法與原來的 pCluster 方法

首先將對本論文提出的方法:pCluster+方法與 Haixun Wang 等人在[5]中所提出的方法進行比較，在圖 5-1 中，屬性數目固定在 30，而資料集內所包含的群數為 0，物件數目則由 1000 到 6000。可以很明顯的看到效能會隨著資料集大小而成正比的改變，但由於並未包含任何的群在其中，因此主要的時間是由產生 MDS 這個動作所決定，但原本的方法中，必需產生 MDSo 及 MDSc，而 pCluster+方法只需產生 MDSc，因此，如同之前所提到的，產生 MDSo 與產生 MDSc 的時間上有著極大的差距，因此導致整個表現上的不同。(圖 5-1 中 pCluster+方法在物件數為 6000 時，需時 1.75 秒左右，但原來的方法需要 161 秒，因此在圖 5-1 中新方法的效能曲線不是那麼明顯。)

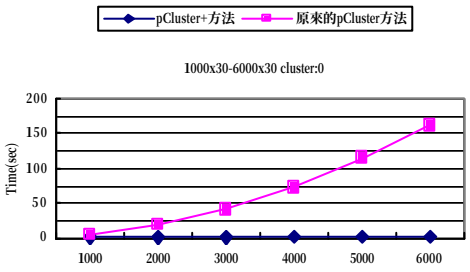

圖 5-1 資料量由 1000 到 6000 ，當 cluster:0

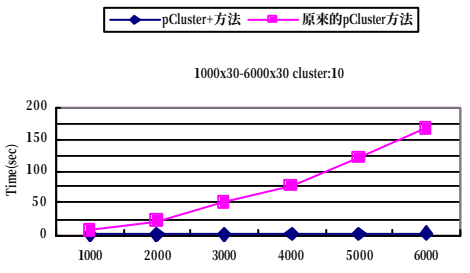

圖 5-2 資料量由 1000 到 6000 ，當 cluster:10

圖 5-2 將資料集內的群數提升為 10 群，其餘則不變，可看到其趨勢大致與圖 5-1 相同，但兩個方法在時間上都略有增加，pCluster+方法在物件數為 6000 時約增加 0.7 秒，而原來的方法則增加了 5 秒。

圖 5-3 裡，物件數目固定為 3000，而屬性維度則由 20 到 100，而資料集包含的群為 0。在圖中可以看出，效能依舊跟資料集大小成正比，但是 pCluster+方法比起前兩張圖的花費要來的高些，原因是 pCluster+方法以 MDSc 為基礎，增加屬性維度，對 pCluster+方法自然具有較大的影響。

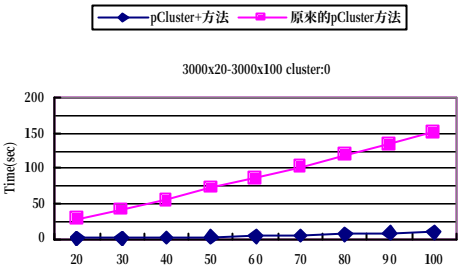

圖 5-3 維度由 20 到 100 ，當 cluster:0

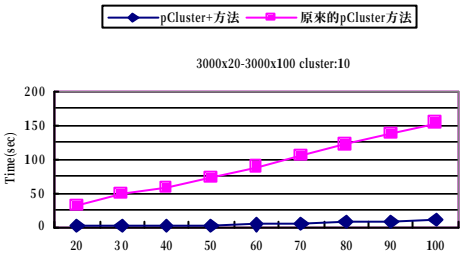

圖 5-4 維度由 20-100 ，當 cluster:10

圖 5-4 的設定與圖 5-3 相同，但包含群數由 0 提升為 10 同樣的，兩個方法都增加了一點時間。從上面可以很清楚的發現，效能與資料集的大小有關，但也可約略看出，資料集中包含的群的多寡也會影響效能表現。

在圖 5-5 中，資料集大小固定在 3000x30，而分別讓資料集裡面所包含的群數由 0 群到 40 群。圖中很清楚的表現了群的多寡影響效能的表現，pCluster+方法亦受影響，0 群時約 1 秒，到了 40 群時則提升時間為 4 秒左右。但在這張圖中也可觀察到一個很有趣的現象。當群數為 20、30 及 40 的時侯，時間突然提高許多。如果仔細回過頭來看看原來的 pCluster 方法，便可以發現問題的所在。在原來 pCluster 方法中，它砍除的部分是一個以 MDSo 及 MDSc 兩兩相砍的迴圈，也就是說當資料集的特性無法讓其在幾回合內就結束這個迴圈，將會導致效能的低落，而圖 5-5 正是這種情況。也就是說資料集內群的多寡，以及

群的特性是否能在一定次數內就結束迴圈，對於整個效能來說其影響將大於資料集的大小。

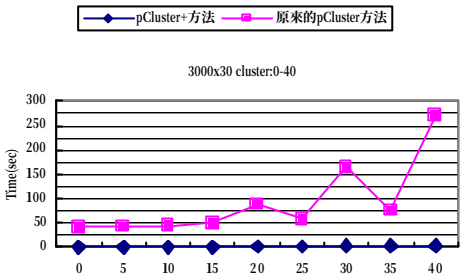

圖 5-5 3000x30 資料集 群數由 0 到 40

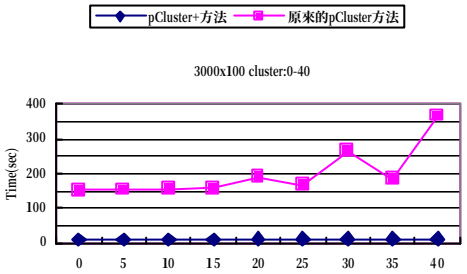

圖 5-6 3000x100 資料集 群數由 0 到 40

接下來的圖 5-6 將屬性維度的數目提升到 100，其餘設定相同，我們發現上述的情形仍舊是存在。總結來說可以歸結出影響所有 pCluster 演算法(包括本論文所提出的方法)效能的因素有三個，資料集的大小，資料集所包含群數的多寡，以及是否"容易"砍除。從上面幾個圖可以看到，pCluster+方法與原來的 pCluster 方法，在效能上差異甚大，以一個 3000x100 的資料集來做進一步分析，便可了解兩者之間的差異圖 5-7。

Dataset 3000 x 100 nr:27 nc:8 t:1 10 clusters

Our new approach for pCluster	Original approach for pCluster
Generate MDSc / MDSo	**Generate MDSc / MDSo**
MDSc num : 789	MDSc num : 789
Time : 9.673 sec.	Time : 9.183 sec.
	MDSo num : 4359
	Time : 146.52 sec.
Prune	**Prune**
Time : 0.371 sec.	Time : 6.309 sec.
Generate cluster	**Generate cluster**
Candidate num : 10	Candidate num : 10
Avg size of candidate : 30 x 10	Avg size of candidate : 30 x 10
Run times : 1	Time : 4.196 sec.
Time : 0.16 sec.	
Total time : 10.204 sec.	**Total time : 166.208 sec.**

圖 5-7 pCluster+方法與 pCluster 方法的比較

從圖 5-7 中右邊是原來的 pCluster 方法，而左邊是 pCluster+方法，從中可以看到，pCluster+方法總共的執行時間為 10 秒左右，而原來的 pCluster 需要約 166 秒，其中最大的差異在於 MDSo 的執行時間，而 pCluster+方法的砍除策略讓我們可以順利的不經由 MDSo 產生出結果，因此節省了這一百多秒，而砍除的策略也較優於原來的砍除策略，至於最後產生 pCluster 的時候，由於原來的方法需要建樹，再進行後序搜尋，而我們則不需，因此比較快。

(二) 資料新增的 pCluster 方法與 pCluster+方法的比較

在這一小節中將對本論文所提出的資料新增的 pCluster 方法進行評估，由於原來的 pCluster 方與 pCluster+方法差距甚大，如果也放在一起比將無法明確的觀察出結果，因此只將資料新增的 pCluster 方法與 pCluster+方法來進行比較，實驗平台則與上一小節一致。

在圖 5-8、5-9 中，以資料集 1000 到資料集 6000 進行實驗，而屬性維度固定為 30，所包含群的數目分別為 0 跟 10。

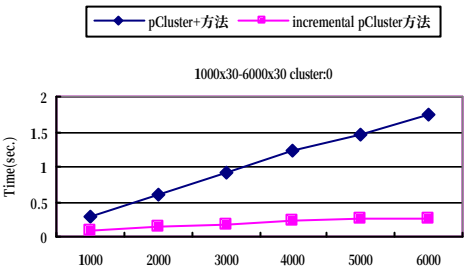

圖 5-8 資料量由 1000 到 6000 ，當 cluster:0

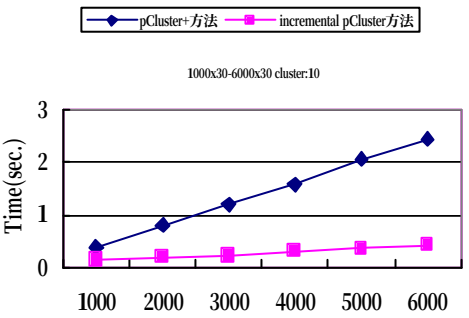

圖 5-9 資料量由 1000 到 6000 ，當 cluster:10

其中可以看出新增資料的 pCluster 方法依然受到資料集大小及包含的群數所影響，但由於執行速度很快，因此包群數的影響只有零點多秒。圖 5-10、5-11 則是將物件數目固定為 3000，屬性維度由 20 到 100，而群數分別為 0 跟 10，比較特別的是 pCluster+方法對於維度的增加較敏感，但新增資料的 pCluster 方法則較不那麼明顯，這是由於演算法間的差異所造成的。

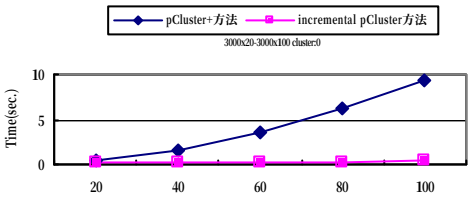

圖 5-10 維度由 20-100 ，當 cluster:0

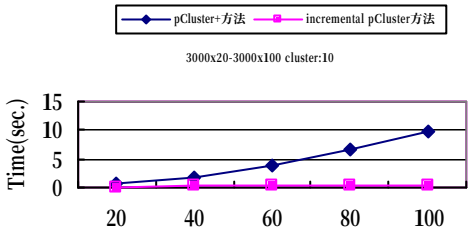

圖 5-11 維度由 20-100 ，當 cluster:10

比較特別的是圖 5-12 的實驗，這個實驗主要是想了解，所加入的物件對於整個分群結果的影響，是否會造成效能上的差異。我們以一個 3000x100 的資料集，其中包含的群數為 10，實驗設計讓具有控制不同群數的物件加入，例如:"c2"是指原來在 2999x100 中會分出 8 群，而新增的物件所控制的群數為 2 群，而執行完新增資料後都會得到 10 群的結果。c4 則是新增物件所控制的群數為 4，依此類推。而實驗的結果如同我們的推測，新增物件所控制的群數會影響個整個演算法的效能。

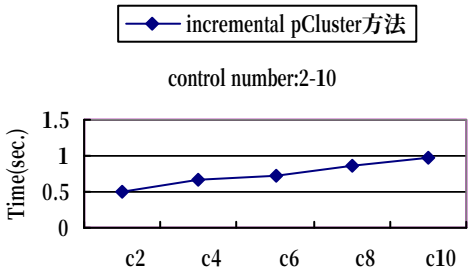

圖 5-12 控制群數 2-10

總結來說，影響新增資料的 pCluster 方法的因素有，資料集的大小、新增物件的控制群數以及是否"容易"砍除(此部分的影響在 pCluster+方法裡已不大，而對於新增資料的 pCluster 方法的影響則更小)。

六、結論

在這篇論文當中所提出的 pCluster+方法，經過實驗的結果，明顯的在執行效能上比原來的 pCluster 方法改進許多，而本論文同時也針對 pCluster 新增資料的問題提出解決的方法，而這也代表 pCluster 能夠被應用到更深更廣的範圍中。當然在 pCluster 中還存在有許多議題值得深入去研究，例如:是否能在資料集中，影響 pCluster 演算法效能的因素，定義出一個確切的關係式? pCluster 是否還能將之擴展，像是可以容錯，或是位移量，pCluster 是否能應用在非數值上呢? 這些問題都非常有趣，值得一一探討。而我們未來的工作將對這些議題做進一步的研究。

參考文獻

[1] C. C. Aggarwal, C. Procopiuc, J. Wolf, P. S. Yu, and J. S. Park. *Fast algorithms for projected clustering.* In SIGMOD, 1999.

[2] C. C. Aggarwal and P. S. Yu. *Finding generalized projected clusters in high dimensional spaces.* In SIGMOD, pages 70–81, 2000.

[3] C. H. Cheng, A. W. Fu, and Y. Zhang. *Entropy-based subspace clustering for mining numerical data.* In SIGKDD, pages 84 – 93, 1999.

[4] H. V. Jagadish, J. Madar, and R. Ng. *Semantic compression and pattern extraction with fascicles.* In VLDB, pages 186 – 196, 1999.

[5] H. Wang, W. Wang, J. Yang, and P. Yu *Clustering by Pattern Similarity in Large Data Sets.* In ACM SIGMOD, 2002.

[6] J. Yang, W. Wang, H. Wang, and P. S. Yu. *δ-clusters: Capturing subspace correlation in a large data set.* In ICDE, 2002.

[7] J. Riedl and J. Konstan. *Movielens dataset*. In http://www.cs.umn.edu/Research/GroupLens

[8] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. *When is nearest neighbors meaningful*. In Proc. of the Int. Conf. Database Theories, pages 217–235, 1999.

[9] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. *Authomatic subspace clustering of high dimensional data for data mining applications.* In SIGMOD, 1998.

[10] S. Tavazoie, J. Hughes, M. Campbell, R. Cho, and G. Church. Yeast micro data set. In http://arep.med.harvard.edu/biclustering/yeast.matrix, 2000.

[11] Y. Cheng and G. Church. *Biclustering of expression data*. In Proc. of 8th International Conference on Intelligent System for Molecular Biology, 2000.