第一章　緒論

1.1 簡介

分群法（Clustering）被定義為在無事前資訊（*a prior* knowledge）下將一個物件的集合（a collection of objects）分類為一群天然群集的集合（a set of natural clusters）[5]。它是一種無外力引導（unsupervised）的資料分類技術。分群法可使得同一群（cluster）中的資料盡可能地相似而不同群中的資料盡可能地相異[1][3]。分群法可用於機器學習（Machine Learning）或資料探勘（Data Mining）的先置處理步驟，也可用於基因體微陣列的分析。

大多數的分群法可分為兩類，階層式（hierarchical）分群法及分割式（partitional）分群法[2][3][4]。其他尚有 Analytic Clustering，Genetic Algorithm，Monothetic Clustering，Graph-Theoretic Clustering，Vector Quantization。

階層式分群法可以產生 nested sequence 並且可以藉由 dendrogram 作圖形式表達，並且不需要指定群數[3]。它不會受到初始分群及局部最小值的影響。階層式分群法可以再細分為聚合式（agglomerative）與分裂式（divisive），聚合式的方式是以一筆資料做為一群，持續地將不同群合併（merging），分裂式的方式是以全體資料做為一群，持續地進行分割（spliting）[4]。

分割式分群法產生一分割，而不是 nested sequence[3]。基於原型的分群法（prototype-based clustering algorithm）是分割式分群法中最受歡迎的

一種。在此類演算法中，每一群可由一原型作為代表。所有的點到原型之距離的總和通常是目標函數（objective function）。這類演算法還可再分為 hard 及 fuzzy 兩種[3]。在 hard clustering 中，每一點都被分配到一個而且是唯一的一個群，其隸屬於該群的程度為 1，在 fuzzy clustering 中每一點都被分配到多個不同的群，其隸屬於各群的程度為 [0,1] 之間正數[3]。

階層式分群法通常是無效率的，且由於它只考慮區域性鄰居因此無法搭配使用全域型態（ global shape ）及群大小（ size of cluster ）的事前資訊。分割式分群法可以搭配使用全域型態及群大小的事前資訊，但是有下述缺點：(1) 難以決定群數 (2) 易受雜訊干擾 (3) 易受初始分群干擾[2]。

1.2 分割式分群法

分割式分群法可被正式定義如下：

給定 n 個從 N 維歐式空間（ N-dimensional Euclidean space ），R^N，的樣本（pattern），決定一分割（partition）將 n 個樣本分至 K 個群中，使得同一群中的樣本相似度高於不同群中的樣本的相似度。

其可被寫為如下的數學形式：

$S=\{x_1,x_2,...x_n\}$ 表示 n 個資料點，$\{C_1,C_2,...C_K\}$ 表示 K 個群。

$C_i \neq \varnothing$ for i= 1,...,K

$C_i \cap C_j = \varnothing$ for i= 1,...,K ， j= 1,...,K, and $i \neq j$

and $\cup_{i=1}^{K} C_i = S$

如何在合理的時間內檢查所有的分割以找出能將前述檢定標準最佳化的解是一重要課題[1]。

如果以窮舉法（ exhaustive enumeration ）解一將 n 個點分至 K 個群中分群問題時，那麼我們必需計算 $\frac{1}{K}\sum_{j=1}^{K}(-1)^{K-j}j^{n}$ 個分割。對 10 點分 2 群的問題我們需要計算$2^{9}-1$個分割，對 50 點分 2 群的問題我們需要計算$2^{50}-1$個分割[1]。

既然列舉所有可能的分割是一組合問題（ combinatorial problem ），heuristics 將會被使用以求能夠在多項式時間（ polynomial time ）內找到良好的解（ good solution)[1]。

第二章　　背景知識

2.1 McQueen's K-means Algorithm

McQueen 的 K-means 演算法中的最初步驟，是以隨機方式將 n 點分至 k 群。可以如下述方式實做 : 隨意取 k 個 seed point，將像第 i 個 seed point 的點分配至群 i，至所有點分配完為止。

在 McQueen 的 K-means 演算法中，每個 pass 中有 n 個 iteration，每個 iteration 有兩個 step。

在 step 1 中，更新群中心，即計算 k 個 cluster 的 centroid。而在 step 2 中，讀取下一筆資料並分配至最近的群中。

當有 n 個連續的 iteration 所有群中心不變時，演算法停止。另外也可以採用連續 n 個 pass 後強迫演算法停止的實作方式。

現欲將 {1,2,3,4,11,12} 分為兩群，執行過程如下。

最初亂數分配為 {1}，{2,3,4,11,12}。

Pass 1

Iteration 1 :
Step 1 : {1}，{2,3,4,11,12}，中心分別為 1 及 6.4
Step 2 : 讀取 1, 距 1 較近, 仍分配在{1}

Iteration 2 :

Step 1 : {1}，{2,3,4,11,12}，中心分別為 1 及 6.4
Step 2 : 讀取 2, 距 1 較近, 改分配, 得 {1,2} {3,4,11,12}

Iteration 3 :
Step 1 : {1,2}，{3,4,11,12}，中心分別為 1.5 及 7.5
Step 2 : 讀取 3, 距 1.5 較近, 改分配, 得 {1,2,3} {4,11,12}

Iteration 4 :
Step 1 : {1,2,3}，{4,11,12}，中心分別為 2 及 9
Step 2 : 讀取 4, 距 2 較近, 改分配, 得 {1,2,3,4} {11,12}

Iteration 5 :
Step 1 : {1,2,3,4}，{11,12}，中心分別為 2.5 及 11.5
Step 2 : 讀取 11, 距 11.5 較近, 不改分配, 得 {1,2,3,4} {11,12}

Iteration 6 :
Step 1 : {1,2,3,4}，{11,12}，中心分別為 2.5 及 11.5
Step 2 : 讀取 12, 距 11.5 較近, 不改分配, 得 {1,2,3,4} {11,12}

Pass 2 :

Iteration 1 :

Step 1 : {1,2,3,4}，{11,12}，中心分別為 2.5 及 11.5

Step 2 : 讀取 1, 距 2.5 較近, 不改分配, 得 {1,2,3,4} {11,12}

Iteration 2 :

Step 1 : {1,2,3,4}，{11,12}，中心分別為 2.5 及 11.5

Step 2 : 讀取 2, 距 2.5 較近, 不改分配, 得 {1,2,3,4} {11,12}

Iteration 3 :

Step 1 : {1,2,3,4}，{11,12}，中心分別為 2.5 及 11.5

Step 2 : 讀取 3, 距 2.5 較近, 不改分配, 得 {1,2,3,4} {11,12}

Iteration 4 :

Step 1 : {1,2,3,4}，{11,12}，中心分別為 2.5 及 11.5

Step 2 : 讀取 4, 距 2.5 較近, 不改分配, 得 {1,2,3,4} {11,12}

Iteration 5 :

Step 1 : {1,2,3,4}，{11,12}，中心分別為 2.5 及 11.5

Step 2 : 讀取 11, 距 11.5 較近, 不改分配, 得 {1,2,3,4} {11,12}

Iteration 1 :

Step 1 : {1,2,3,4}，{11,12}，中心分別為 2.5 及 11.5

Step 2 : 讀取 12, 距 11.5 較近, 不改分配, 得 {1,2,3,4} {11,12}

由於連續 6 個 iteration 不改分配，演算法停止。

2.2 Forgy's K-means Algorithm

Forgy 版的 K-means 演算法和 McQueen 版大致相同，差別在於讀完所有點後才一次重分配所有點並重新計算群中心。

現欲將 {1,2,3,4,11,12} 分為兩群，執行過程如下。

Pass 1 :

Step 1 : {1}，{2,3,4,11,12}，中心分別為 1 及 6.4

Step 2 : 重分配所有點得 {1,2,3}，{4,11,12}

Pass 2 :

Step 1 : {1,2,3}，{4,11,12}，中心分別為 2 及 9

Step 2 : 重分配所有點得 {1,2,3,4}，{11,12}

Pass 3 :

Step 1 : {1,2,3,4}，{11,12}，中心分別為 2.5 及 11.5

Step 2 : 重分配所有點得 {1,2,3,4}，{11,12}

由於連續 2 個 pass 不改分配，演算法停止。

McQueen 及 Forgy 的版本均必然收斂，因所有的分配法是有限的，而 total sum of within-cluster square-error-sum 在每分配資料點時，都必然下降，且出現過的 partition 絕不再出現，隨著 iteration (pass) 前進，系統必定停止。

McQueen 及 Forgy 的版本雖然都必定收斂，但是很容易在 total sum of within-cluster square-error-sum 掉入 local minimum 時停止。

McQueen 及 Forgy 的共同優點在於 (1) 易實作 (2) 必定收斂 (3) 對給定的 initialization 時間不會太長。它們的共同缺點在於 (1) 不同的 initialization 很可能結果不同 (2) 需試很多不同的 initialization 後才知道要採用何組數據 (3) 很容易受到 noise data 影響。

2.3 Hierarchical Methods

Hierarchical methods 又細分為兩種，聚合式（ agglomerative ）與分裂式（ divisive ）。

Hierarchically divisive clustering 的詳細步驟如下 ：

Step 1 ： 將所有資料視為一個 cluster

Step 2 ： 在接下來的每一步驟中，依據某種 rule，從現有 clusters 挑選出最該分裂的 cluster

Step 3 ： 重覆 step 2 直到 (1) 產生所要的群數 或 (2) 每一資料點就是一群

實例如下 ：

Step 1 ： {1,3,5,6,78,79,96,97,98}

Step 2 ： {1,3,5,6}{78,79,96,97,98}

Step 3 ： {1,3,5,6}{78,79}{96,97,98}

Step 4 ： {1,3}{5,6}{78,79}{96,97,98}

Hierarchically agglomerative clustering 的詳細步驟如下 ：

Step 1 ： 將每一資料點視為一個 cluster

Step 2 ： 在接下來的每一步驟中，每次 merge 兩個最近的 cluster

Step 3 ： 重覆 step 2 直到 (1) 產生所要的群數 或 (2) 所有資料合併為一個 cluster 時

實例如下 ：

Step 1 : {1}{3}{5}{6}{78}{79}{96}{97}{98}

Step 2 : {1}{3}{5,6}{78}{79}{96}{97}{98}

Step 3 : {1}{3}{5,6}{78,79}{96}{97}{98}

Step 4 : {1}{3}{5,6}{78,79}{96,97}{98}

Step 5 : {1}{3}{5,6}{78,79}{96,97,98}

Step 6 : {1,3}{5,6}{78,79}{96,97,98}

Step 7 : {1,3,5,6}{78,79}{96,97,98}

Step 8 : {1,3,5,6}{78,79,96,97,98}

Step 9 : {1,3,5,6,78,79,96,97,98}

其中兩群 A,B 間的距離，有以下幾種常見方式 ：

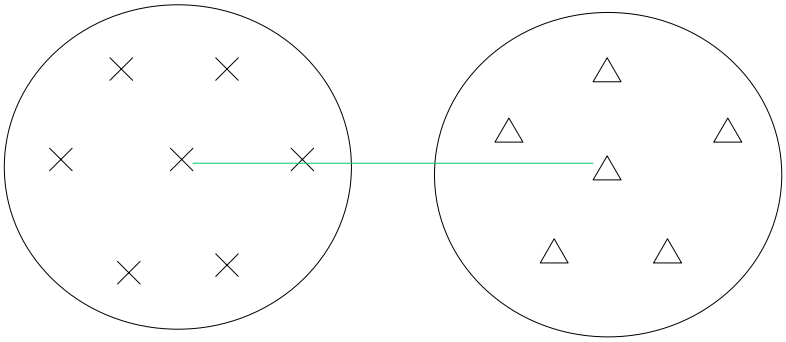

圖 2.1

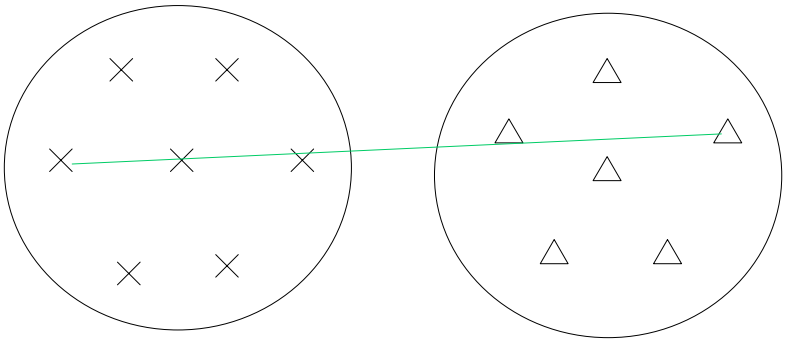

圖 2.2

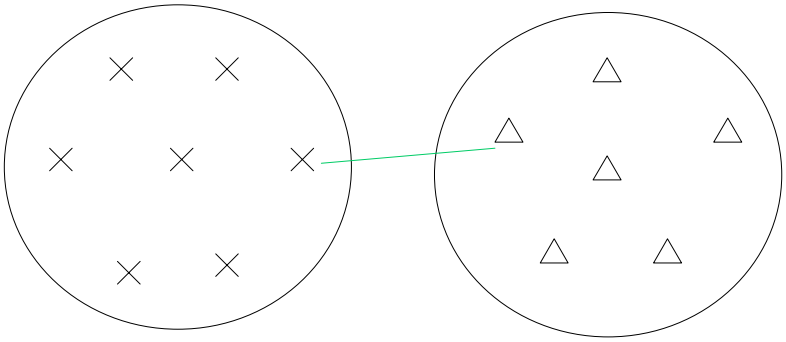

圖 2.3

(1)

$D_{mean}(A,B) = d(\bar{a},\bar{b})$

$\bar{a},\bar{b}$ 為 A,B 的群中心, 又稱為 Centroid-Linkage Method

如圖 2.1

(2)

$D_{\max}(A,B) = \max_{a\in A,b\in B}(a,b)$, 又稱為 Complete-Linkage Method

如圖 2.2

(3)

$D_{\min}(A,B) = \min_{a\in A,b\in B}(a,b)$

如圖 2.3, 又稱為 Single-Linkage Method

(4)

$D_{average}(A,B) = \frac{1}{|A||B|}\sum_{a\in A,b\in B}(a,b)$

聚合式（ agglomerative ）與分裂式（ divisive ）的 hierarchical clustering 都不適用於大量資料，執行速度緩慢，很耗費記憶體。

2.4 Peak-Climbing Clustering

Peak-climbing clustering 是一種 automatic, histogram-based 的分群演算法。有時也稱為 Mode-seeking clustering 或是稱為 valley-seeking clustering。

Peak-climbing clustering 的詳細步驟如下 ：

Step 1： 將一 2-dimensional 的資料群切割為 $N \times N$ 的 cell，並統計每 cell 內有多少資料點

Step 2： 檢查每一 cell 的 8 個鄰居，向點數最多者畫箭頭。若點數最多的鄰居點數少於自己則不畫

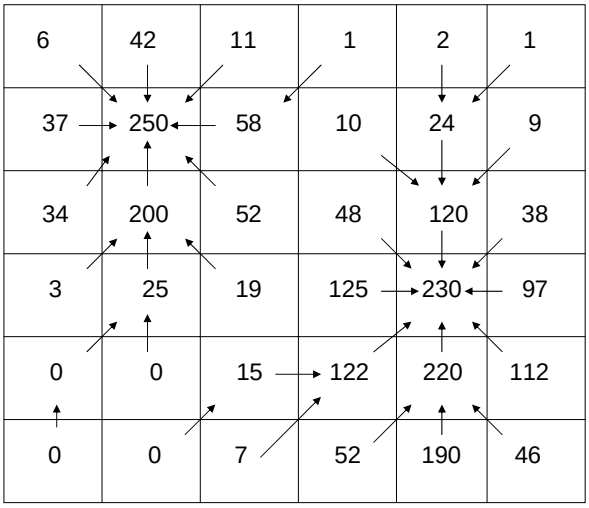

圖 2.4

Peak-climbing 的缺點在於 (1) cell 的 size 的決定，太多 cell 得到太多群，太少 cell 可能只得到 1 群。(2)不適合更高維度的資料。

2.5 Fuzzy K-means Clustering

在 fuzzy clustering 中每一點都被分配到多個不同的群。其隸屬於各群的程度為 [0,1] 之間正數。優點是能給予更多關於資料的結構的資訊，缺點是結果報告過多而不易閱讀，演算法較耗時。

符號 ：

$\{x_j\}_1^n$ 是 n 個輸入資料點

$\{v_j\}_1^k$ 是 k 個群中心

q 是控制參數

u_{ij} 是點 x_j 在群的 i 機率，其中

$0 \le u_{ij} \le 1, \forall i \forall j$

$u_{1j} + u_{2j} + \cdots + u_{kj} = 1$

目的 ：

$$\min_{\{u_{ij}\}\{v_i\}} \sum_{i=1}^{k} \sum_{j=}^{n} (u_{ij})^q \| x_j - v_i \|^2$$

Fuzzy K-means clustering 的詳細步驟如下 ：

Step 1 ： 隨機取 k 個值 $\{v_1, v_2, \cdots, v_k\}$ 做為群中心

Step 2 ： 計算 membership coefficient

$$u_{ij} = \frac{(\| x_j - v_i \|^{-2})^{\frac{1}{q-1}}}{\sum_{l=1}^{k} (\| x_j - v_l \|^{-2})^{\frac{1}{q-1}}} = \frac{1}{\sum_{l=1}^{k} (\frac{d_{ji}}{d_{jl}})^{\frac{2}{q-1}}}$$

Step 3 ： 重新計算新的群中心

$$v_i^{new} = \frac{\sum_{j=1}^{n} (u_{ij})^q x_j}{\sum_{j=1}^{n} (u_{ij})^q}$$

Step 4 : 重覆 step 2, step 3 直到

$$\max_{ij} |u_{ij} - u_{ij}^{new}| < \varepsilon$$

雖然目的是 minimization，但通常所得到的結果只是 local-minimum，如同 K-means 一樣。

控制參數 q 設定為大於 1 的值，通常是 2，使程序能收斂。

2.6 Simulated Annealing

模擬退火法（Simulated Annealing）的目的在於在從所有 partition 中選取使系統能量最小的 partition。

符號 :

δ : 是一個 perturbation rule

T : 溫度（為一控制參數）

MaxIt : Max Number of Iteration（為一給定的參數）

α : 降溫倍率

T_0 : 初始溫度

T_{final} : 最終溫度

Simulated annealing clustering 的詳細步驟如下 :

Initialization :

$T \leftarrow T_0$

任取一 partition

Step 1 : For i← 1 to MaxIt do

$P' \leftarrow \delta(P)$

$\Delta \equiv E(P') - E(P)$

接受 P' 為新的 P ，當 $\Delta < 0$

接受 P' 為新的 P ，當 $\Delta > 0$ 且 $e^{\frac{-\Delta}{T}} \geq$(新選出的亂數)

Step 2 : $T \leftarrow \alpha T$

Step 3 : goto step 2 unless $T < T_{final}$

溫度越高越容易接受新的 partition。而［接受 P' 為新的 P ，當 $\Delta > 0$ 且 $e^{\frac{-\Delta}{T}} \geq$(新選出的亂數)］的步驟是用於避開落入 local-minimum。

2.7 Deterministic Annealing

符號 :

y_i : 第 i 群的群中心

$E_{x,i} = | x - y_i |^2$

$\beta = \frac{1}{T}$

$$0 \leq u_{x,i} = (u_{x,i})|_\beta = \frac{e^{-\beta E_{x,i}}}{\sum_{l=1}^{k} e^{-\beta e_{x,l}}} \qquad (2.7.1)$$

將（2.7.1）式對 y_i 微分後得

$$y_i^{new} = \left(\sum_x x \frac{e^{-\beta E_{x,i}}}{\sum_{l=1}^{k} e^{-\beta e_{x,l}}}\right) / \left(\sum_x \frac{e^{-\beta E_{x,i}}}{\sum_{l=1}^{k} e^{-\beta e_{x,l}}}\right) \qquad (2.7.2)$$

(2.7.2) 式無法直接解，必須以 iterative model 從舊的群中心求新的群中心。

即猜 $\{y_i^{initial}\}_{i=1}^{k}$，然後更新為 $\{y_i^{(1)}\}_{i=1}^{k}$，再由 $\{y_i^{(1)}\}_{i=1}^{k}$ 得到 $\{y_i^{(2)}\}_{i=1}^{k}$，以此類推。

Deterministic annealing clustering 的詳細步驟如下 :

Initialization : $\beta = 0^+$，隨機取 k 個值 $\{y_1, y_2, \cdots, y_k\}$ 做為群中心

Step 1 : 依 (2.7.2)更新群中心，直到 $\{y_1, y_2, \cdots, y_k\}$ 穩定 (stable)

Step 2 : 增加 β

Step 3 : goto step 1 unless $\beta \geq \beta_{final}$

當 $\beta = 0 \Leftrightarrow$ high temperature $\Leftrightarrow$ high fuzziness; 反之，當 $\beta = \infty \Leftrightarrow$ low temperature $\Leftrightarrow$ hard clustering。

Deterministic Annealing 和 Simulated Annealing 相同，可避開 K-means 會落入 local-minimum 的缺點。

第三章　機率式重分配的模擬退火 K-means 演算法

3.1　簡介

2001 年時 Maulik 及 Pakhira 提出了一個有效率的分割式分群法（ partitional clustering), Simulated-Annealing K-means Clustering with Probabilistic Redistribution, 簡稱 SAKM-clustering。這個方法整合了模擬退火法取得系統最小能量及 K-means 快速搜尋的能力。它適用於搜尋多維特徵空間中適當分群並使得分群結果在相似度測度上最佳化。

3.2　SAKM 所使用的 K-means 演算法

在 SAKM 中所使用的 K-means 演算法是屬於 Frogy’s method，其內容詳列如下。

Step 1 : 以亂數從 n 個資料點 $\{x_1, x_2, \ldots, x_n\}$ 中選取 K 個初始群中心 $\{z_1, z_2, \ldots, z_k\}$。

Step 2 : 將點 x_i , i=1, 2, . . . , n , 分配到第 j 個群, $j \in \{1,2,\ldots,K\}$ 若且唯若

$\| x_i - z_j \| < \| x_i - z_p \|$, p=1, 2, …, K 且 j≠p。

Step 3 : 計算新的群中心 $z_1^*, z_2^*, \ldots, z_K^*$ 如下

$$z_i^* = \frac{1}{n} \sum_{x_j \in C_i} x_j \quad ,$$

$i \in \{1,2,\ldots,K\}$

Step 4 : 如果對所有 $i \in \{1,2,\ldots,K\}$ 而言 $z_i^* = z_i$, 演算法停止。否則跳至 Step 2 。

當 Step 4 不會正常停止時， 我們會讓程式在執行一固定的最大執行次數後停止。

3.3　Metropolis 演算法

退火法（ Annealing ）是一個將晶體（ crystal ）從液態降溫至固態的物理過程。如果降溫的過程是緩慢的，那麼最終系統可非常趨近於最低能量狀態。這樣的降溫過程可用 Metropolis 演算法模擬[1]。

Metropolis 演算法中，現在的系統組態為 C_i，其能量為 E_i，那麼可以藉由對 C_i 作微擾（ small perturbation ）而得到次一組態為 C_j，且其能量為 E_j。當 $E_j \le E_i$ 時，C_j 被接受成為新的組態，否則 C_j 將有 $\exp(-\frac{E_j - E_i}{k_B T})$ 的機率被接受成為新的組態。其中 T 是絕對溫度，k_B 是 Boltzmann 常數[1][6]。如果降溫的過程是夠緩慢的，那麼晶體在每一溫度都可達到熱平衡。在 Metropolis 演算法中是藉由在每一溫度時進行足夠多次的微擾來達成。

3.4 SAKM-clustering 中的重分配機制

在 SAKM-clustering 中， 對系統的微擾是藉由重分配適當的資料點來達成。首先所有的資料點是被隨機分配到 K（ K 事先給定 ）個群中，再計算群中心。每一個資料點和其群中心的相關度是和該點到其群中心的 Euclidean distance 成平方反比，所以每一個群中離群中心最遠的資料點是重分配的最適當選擇。我們會以

$$\exp(-\frac{[D_{ik} - D_{ij}]}{T_t}) \qquad (1)$$

的機率重分配該資料點，其中 $[x]+ = \max(x,0)$，$D_{ik} = \| x_i - z_k \|$，$k \neq j$ 。 T_t 是一溫度控制程序（temperature schedule），其必須符合 $T_1 \ge T_2 \ge \ldots T_t = 0 \quad (\lim t \to \infty)$ 的性質。t 是退火程序中降溫的次數。

3.5 SAKM-clustering 中的 temperature schedule

採用 $T_t = T_1/(1+\ln t)$ $(\lim t \to \infty)$ 形式的 logarithmic annealing schedule 可以確保模擬退火程序的漸近式收斂，其中 $t \geq 1$ 而 T_1 是初始溫度。然而實際上 logarithmic annealing 是非常慢的，因此我們會採用 geometric schedule，其形式為 $T_t = (1-\alpha)^t * T_1$，其中 α 是一非常靠近 0 的正實數。當 $T_t \to 0$，將不會有任何擾動，而終止條件也假設已達到了。實際上在此之前系統組態已凍結住了。

3.6 SAKM-clustering 演算法

SAKM 分群演算法內容如下。

1. $T = T_{\max}$
2. 將資料隨機分配至 K 個群中以形成初始系統組態 C，及其系統能量 E。
3. while $T > T_{\max}$
4. for i=0 to N_T do
5. 依據 equation (1) 將 C 中的資料重分配以取得 C' 及 E' 。
6. 如果 $(E'-E) \leq 0$ ， C ← C'
7. 否則將有 $\exp(-\frac{E_j - E_i}{k_B T})$ 的機率 C ← C'
8. end for
9. 減少 T
10. end while

第四章　　系統實作

4.1 變數與參數設定

在執行過程中，資料點點數及所要分群的群數是由使用者給定，隨即以亂數方式取得所需的資料點內容。這些資料點將被使用於 SAKM 及 K-means 演算法中。

初始溫度與最終溫度是由使用者給定，是僅用於 SAKM 演算法中的變數。$\alpha = 0.02$，$N_T = 3$，都是已固定於程式中的系統參數。

在 K-means 中，為預防程式不會停止，一般會設一變數，當程式執行迴圈的次數到此一定值時將停止。我們已在程式中將其設定為 100。

4.2 操作介面與流程

操作介面分為四部份，如圖 4.1。左上部份為資料點點數 num（ 預設 1000 點 ）設定及群數 cnum（ 預設 5 點，且不能超過 32 ）設定，SAKM 系統最初溫度 Temp.（ 預設 300°K ）與最末溫度 Min Temp.（ 預設 10°K ）設定，進行資料重分配是否即時更新畫面的選項欄 update（ 預設為是即時更新畫面 ），初始化執行鍵 Build，資料點詳細內容。左下部份及右下部份分別是 SAKM 及 K-means 執行結果的顯示畫面。右上部份為 SAKM 及 K-means 執行所得的 total sum of within-cluster square-error-sum 的比較。

現欲將 100 點分 4 群，SAKM 初溫與末溫為 100°K 與 10°K。首先在 num 一欄設為 100，在 cnum 一欄設為 4，在 Temp. 一欄設為 100，在 Min Temp. 一欄設為 10，如圖 4.2。接下來按下 Build 鍵，得到初始化結果，如圖 4.3

接下來按下 toolbar 上的“執行”，所得結果如圖 4.4。其中在右上部份有兩條曲線分別是 SAKM 及 K-means 執行結果，另外有四個標記，分別為。”3822.066895 #KM,0”，”3737.180908 #KM,5”，”3462.127441 #SAKM,0”，”2200.310791 #SAKM, 113”。分別代表以 K-means 初使化時系統 total sum of within-cluster square-error-sum 為 3822.066895，K-means 終

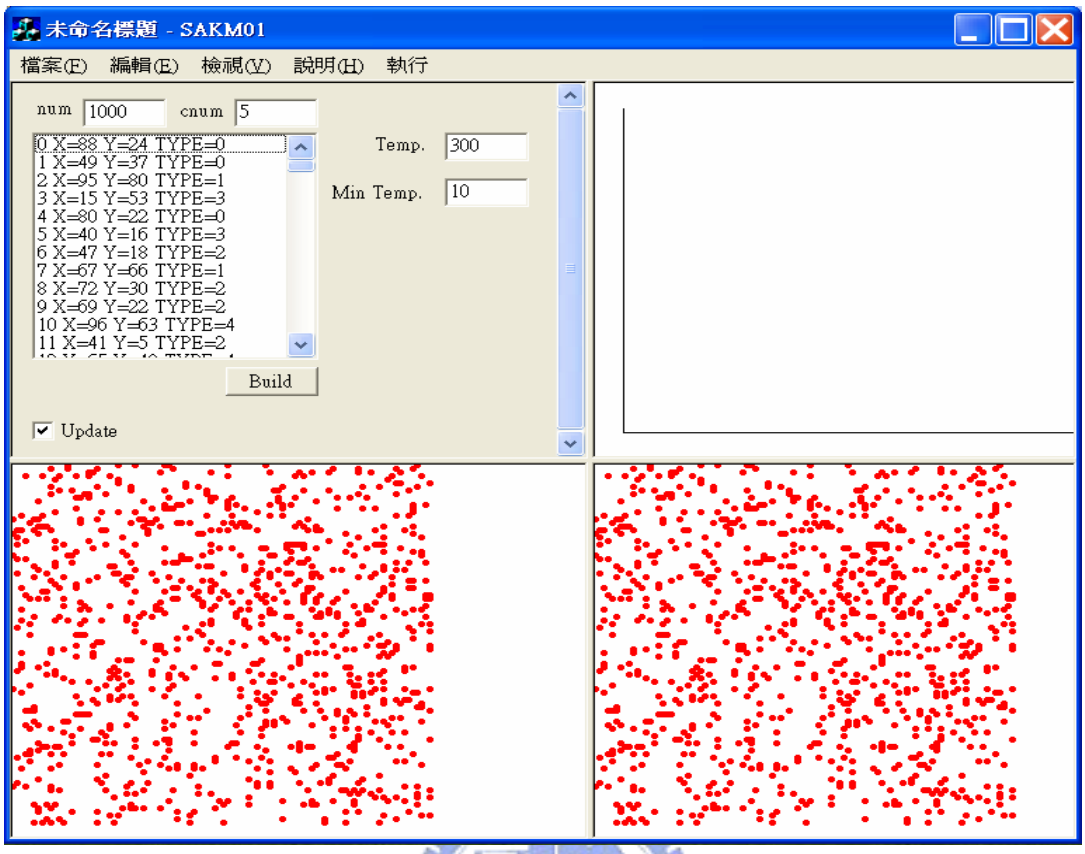
未命名標題 - SAKM01
檔案(F) 編輯(E) 檢視(V) 說明(H) 執行
num 1000
cnum 5
0 X=88 Y=24 TYPE=0
1 X=49 Y=37 TYPE=0
2 X=95 Y=80 TYPE=1
3 X=15 Y=53 TYPE=3
4 X=80 Y=22 TYPE=0
5 X=40 Y=16 TYPE=3
6 X=47 Y=18 TYPE=2
7 X=67 Y=66 TYPE=1
8 X=72 Y=30 TYPE=2
9 X=69 Y=22 TYPE=2
10 X=96 Y=63 TYPE=4
11 X=41 Y=5 TYPE=2
Temp. 300
Min Temp. 10
Build
Update

圖 4.1

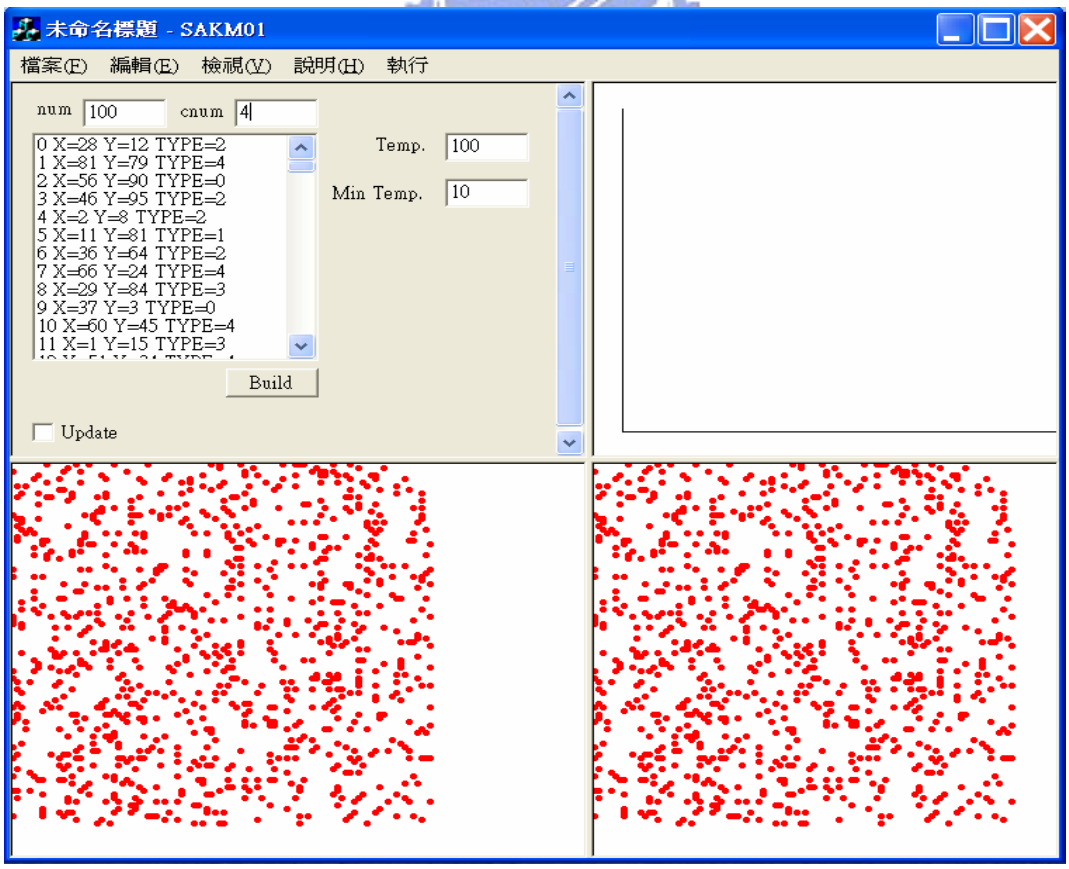
未命名標題 - SAKM01
檔案(F) 編輯(E) 檢視(V) 說明(H) 執行
num 100
cnum 4
0 X=28 Y=12 TYPE=2
1 X=81 Y=79 TYPE=4
2 X=56 Y=90 TYPE=0
3 X=46 Y=95 TYPE=2
4 X=2 Y=8 TYPE=2
5 X=11 Y=81 TYPE=1
6 X=36 Y=64 TYPE=2
7 X=66 Y=24 TYPE=4
8 X=29 Y=84 TYPE=3
9 X=37 Y=3 TYPE=0
10 X=60 Y=45 TYPE=4
11 X=1 Y=15 TYPE=3
Temp. 100
Min Temp. 10
Build
Update

圖 4.2

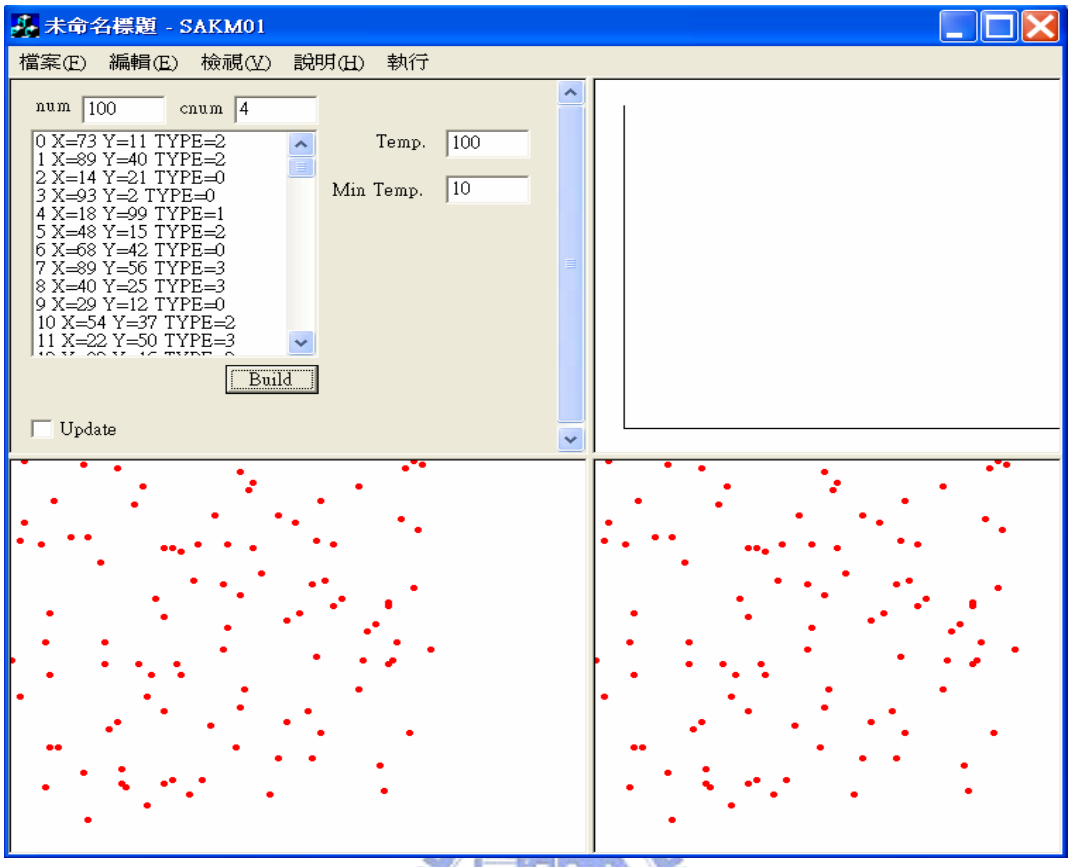
未命名標題 - SAKM01
檔案(F) 編輯(E) 檢視(V) 說明(H) 執行
num 100
cnum 4
0 X=73 Y=11 TYPE=2
1 X=89 Y=40 TYPE=2
2 X=14 Y=21 TYPE=0
3 X=93 Y=2 TYPE=0
4 X=18 Y=99 TYPE=1
5 X=48 Y=15 TYPE=2
6 X=68 Y=42 TYPE=0
7 X=89 Y=56 TYPE=3
8 X=40 Y=25 TYPE=3
9 X=29 Y=12 TYPE=0
10 X=54 Y=37 TYPE=2
11 X=22 Y=50 TYPE=3
Temp. 100
Min Temp. 10
Build
Update

圖 4.3

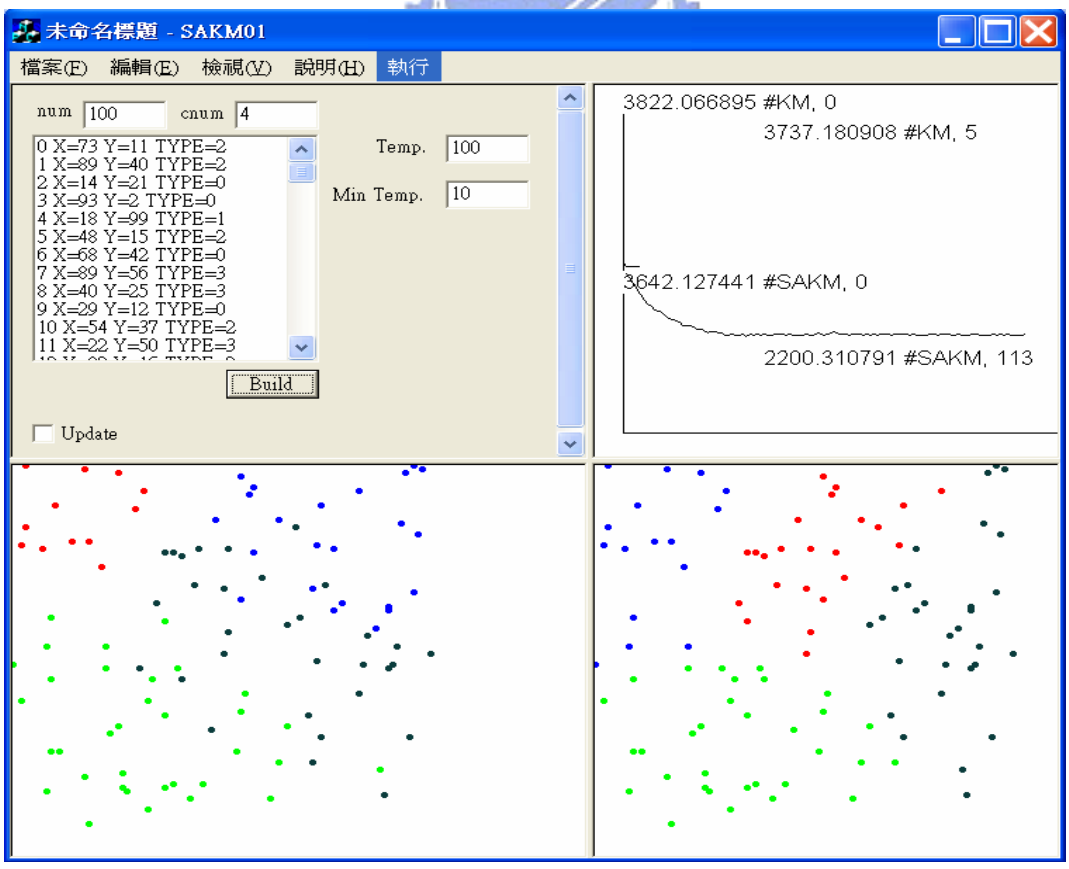
未命名標題 - SAKM01
檔案(F) 編輯(E) 檢視(V) 說明(H) 執行
num 100
cnum 4
0 X=73 Y=11 TYPE=2
1 X=89 Y=40 TYPE=2
2 X=14 Y=21 TYPE=0
3 X=93 Y=2 TYPE=0
4 X=18 Y=99 TYPE=1
5 X=48 Y=15 TYPE=2
6 X=68 Y=42 TYPE=0
7 X=89 Y=56 TYPE=3
8 X=40 Y=25 TYPE=3
9 X=29 Y=12 TYPE=0
10 X=54 Y=37 TYPE=2
11 X=22 Y=50 TYPE=3
Temp. 100
Min Temp. 10
Build
Update
3822.066895 #KM, 0
3737.180908 #KM, 5
3642.127441 #SAKM, 0
2200.310791 #SAKM, 113

圖 4.4

止時共經過 5 次 iteration 且其 total sum of within-cluster square-error-sum 為 3737.180907，以 SAKM 初使化時系統 total sum of within-cluster square-error-sum 為 3462.127441，SAKM 終止時共經過 113 次 iteration 且其 total sum of within-cluster square-error-sum 為 2200.310791。

第五章　　SAKM 與 K-means 的效能比較

5.1　　點數固定時的比較

不管是少量資料（ 如圖 5.1 ）或大量資料（ 如圖 5.2 ），當點數固定時 SAKM 的最初 TSSE（ Total sum of square error, 即 total sum of within-cluster square-error-sum ）比 K-means（ KM ）的最初 TSSE 來的少，SAKM 的最末 TSSE 也比 K-means（ KM ）的最末 TSSE 來的少。SAKM 中 TSSE 的下降幅度大都超過 50%，而同一組資料 KM 中 TSSE 的下降幅度大都很微小，通常是 2%到 5%。通常 SAKM 的執行次數是 KM 執行次數的 5 到 10 倍，如圖 5.3 及圖 5.4。

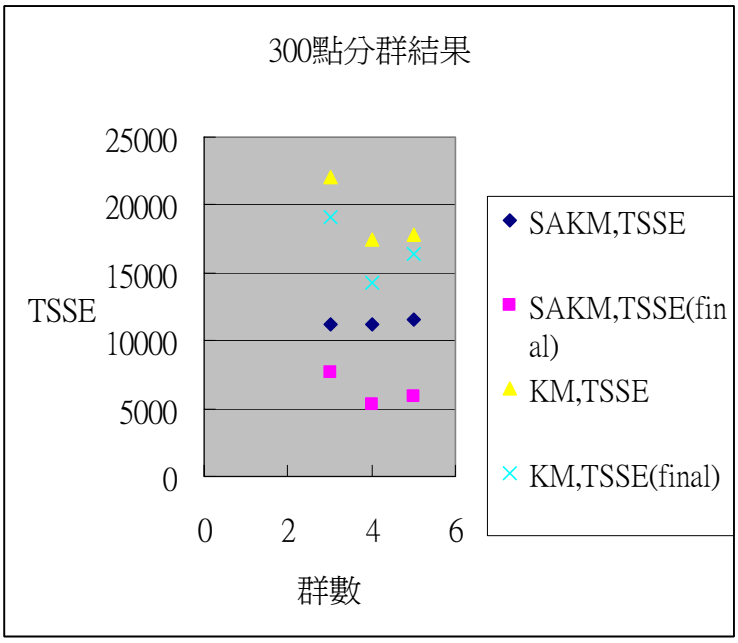

圖 5.1

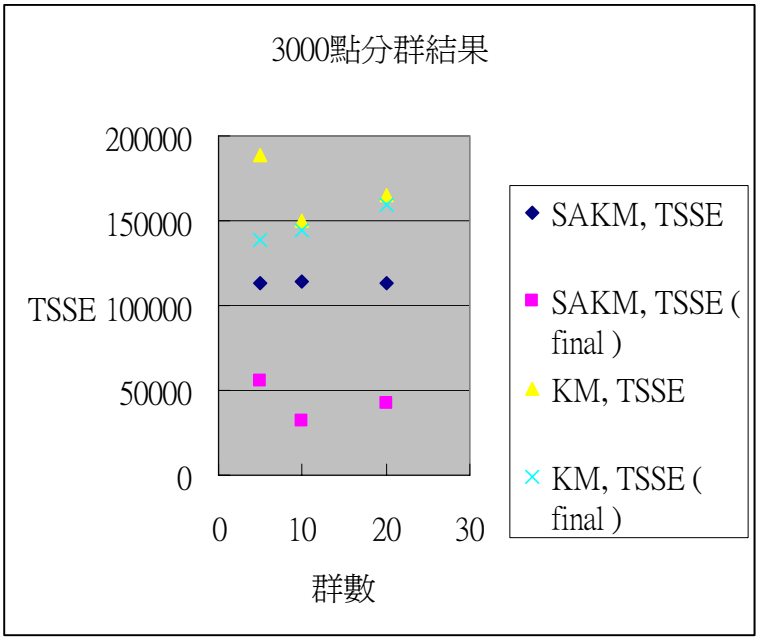
3000點分群結果
200000
150000
TSSE 100000
50000
0
0
10
20
30
群數
SAKM, TSSE
SAKM, TSSE (final)
KM, TSSE
KM, TSSE (final)

圖 5.2

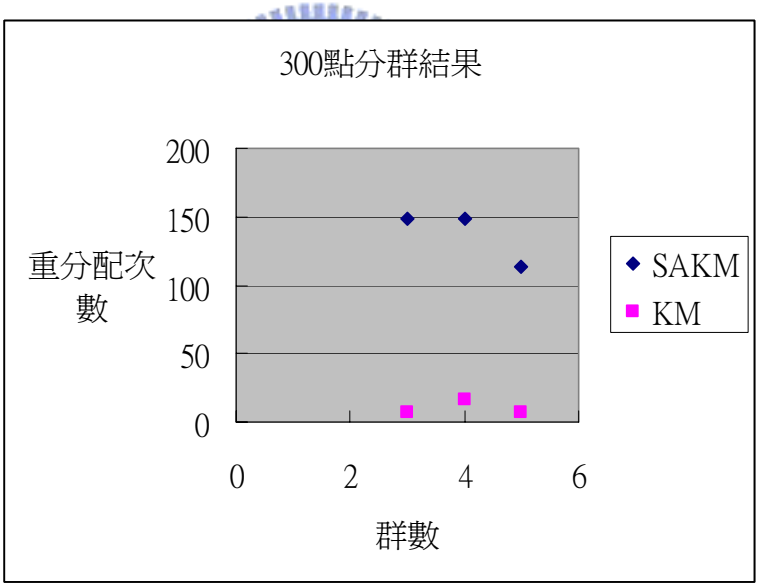
300點分群結果
200
150
重分配次數
100
50
0
0
2
4
6
群數
SAKM
KM

圖 5.3

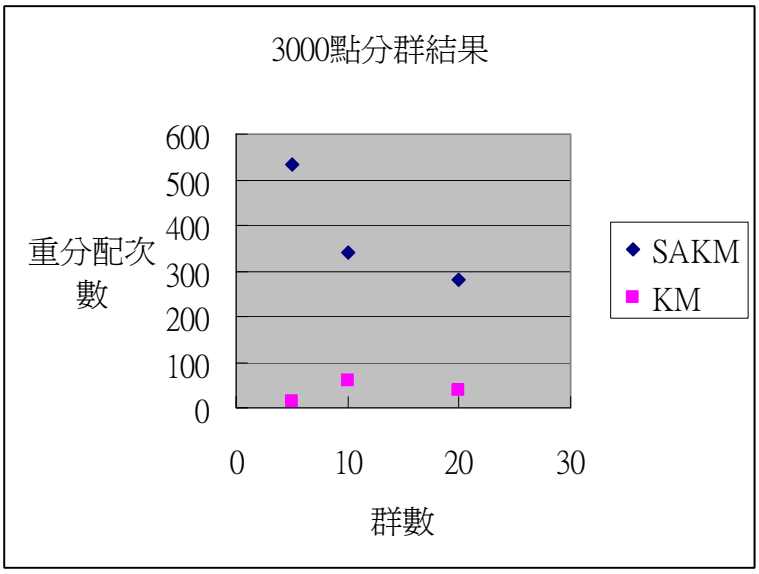

圖 5.4

5.2 分群數固定時的比較

不管是少量資料（如圖 5.5）或大量資料（如圖 5.6），當分群數固定時 SAKM 的最初 TSSE 比 K-means（KM）的最初 TSSE 來的少，當資料量很大時幾乎是 1 比 2。SAKM 的最末 TSSE 也比 K-means（KM）的最末 TSSE 來的少。SAKM 中 TSSE 的下降幅度大都超過 50%，而同一組資料 KM 中 TSSE 的下降幅度大都很小。通常 SAKM 的執行次數是 KM 執行次數的 5 到 10 倍，如圖 5.7 及圖 5.8。

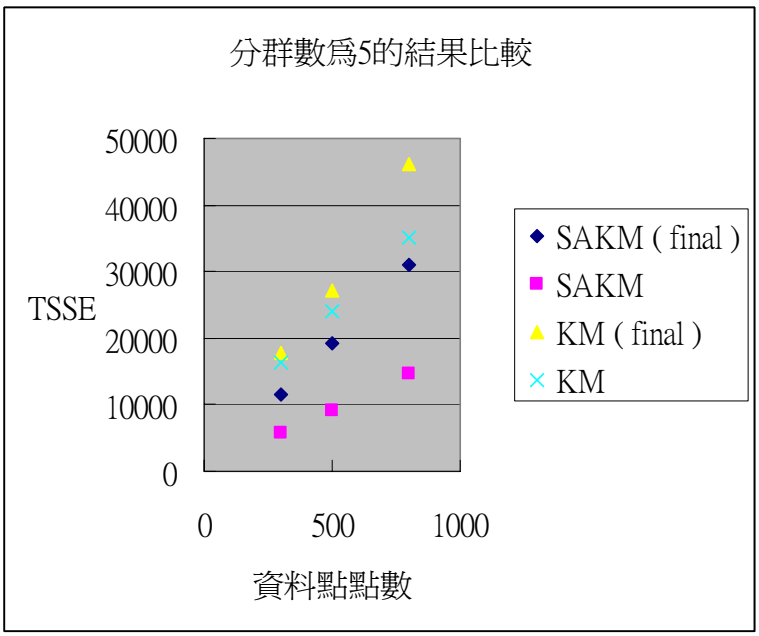

圖 5.5

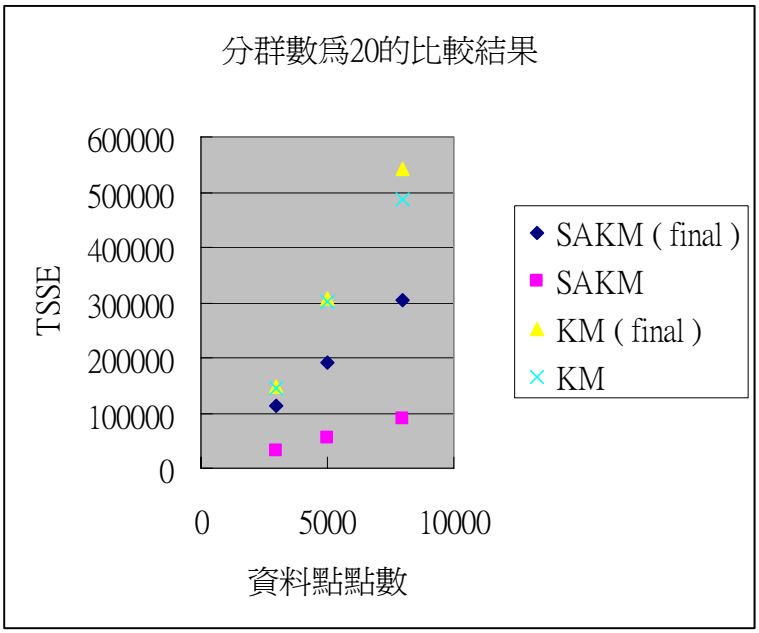
分群數爲20的比較結果
600000
500000
400000
300000
200000
100000
0
TSSE
0
5000
10000
資料點點數
SAKM (final)
SAKM
KM (final)
KM

圖 5.6

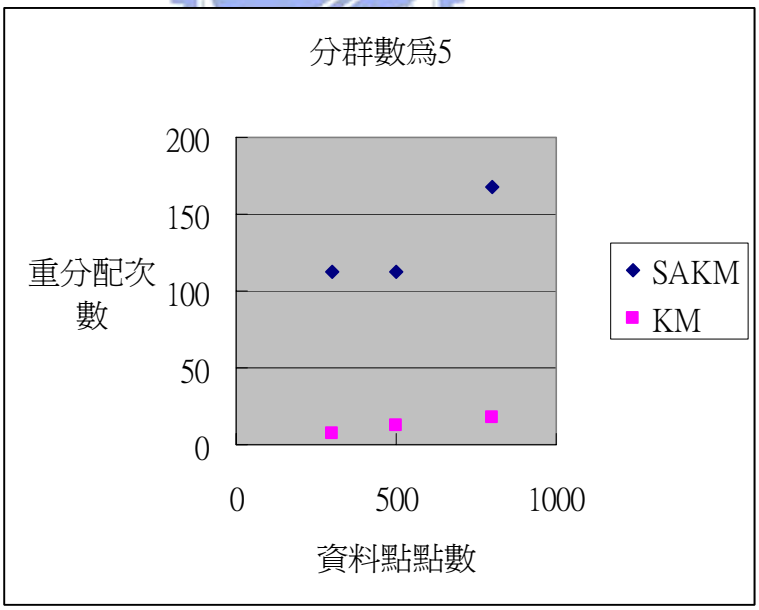
分群數爲5
200
150
100
50
0
重分配次數
0
500
1000
資料點點數
SAKM
KM

圖 5.7

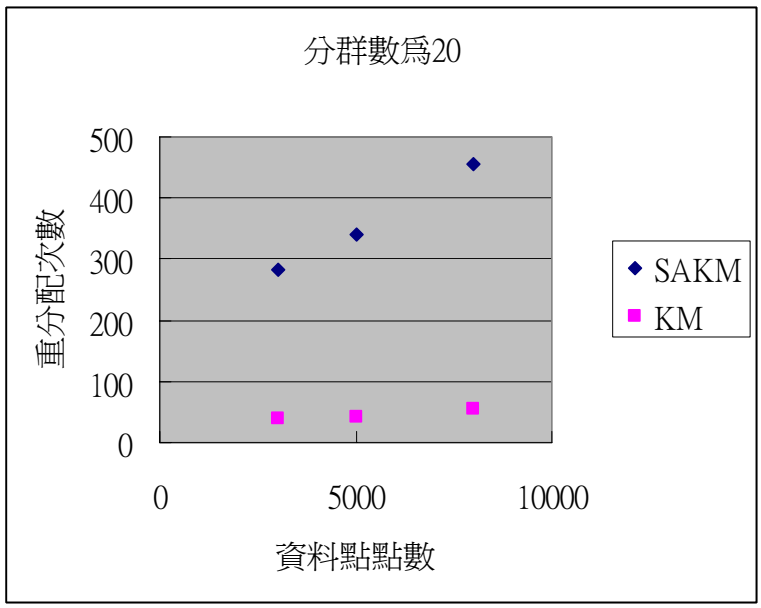

圖 5.8

5.3 SAKM 中點數固定時分群數與溫度的關係

SAKM 中點數固定時分群數與溫度的關係如圖 5.9。群數越多初始溫度越低。

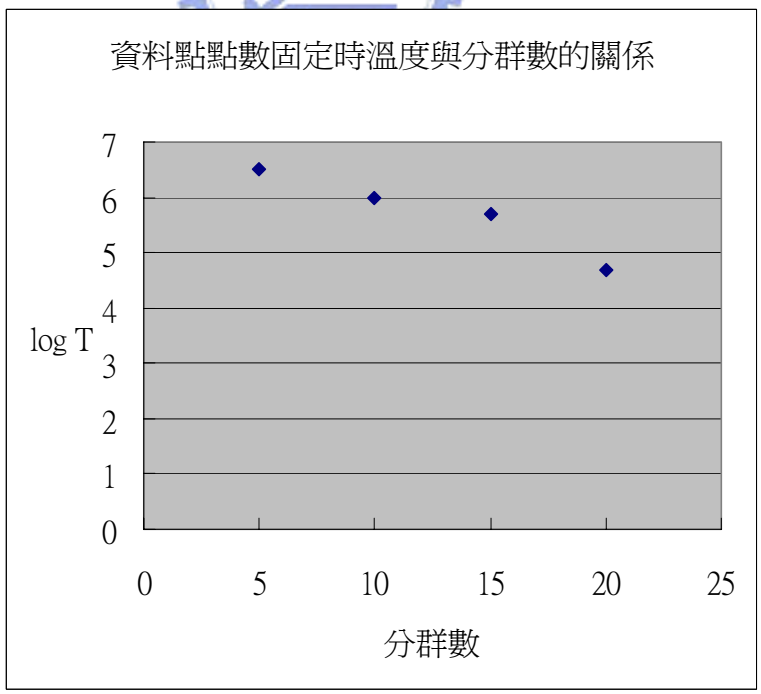

圖 5.9

5.4 SAKM 中分群數固定時點數與溫度的關係

SAKM 中分群數固定時點數與溫度的關係是呈 piecewise exponential 的關係，如圖 5.10 是分群數為 5 時點數與溫度的關係。以 1000 點為界，少於 1000 點時是較快的指數函數，大於 1000 點時是較慢的指數函數。以分群數為 5 時為例，

$f(n) \cong e^{0.086995n-14.5885}, 300 \le n \le 800$ ，

$f(n) \cong 100000e^{0.00230258n}, 1000 \le n \le 5000$ 。

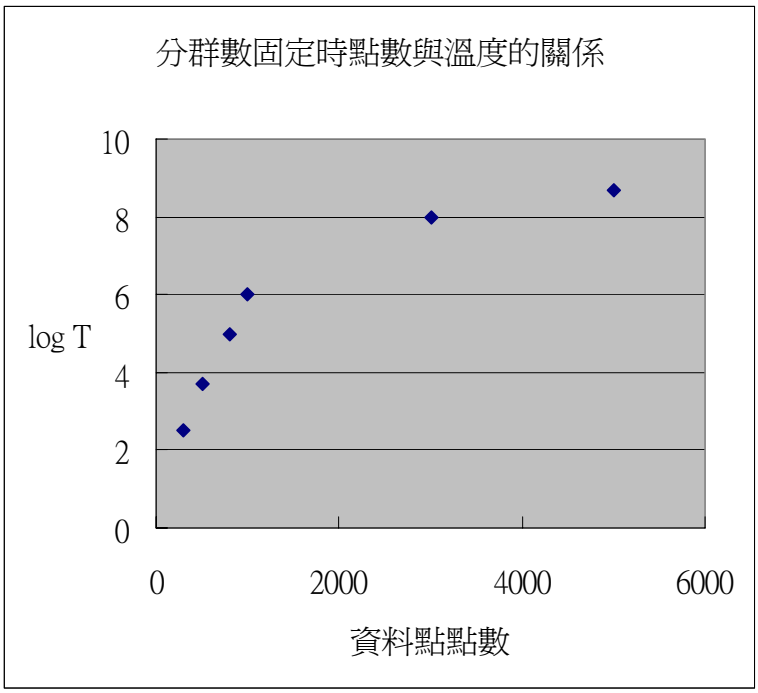

圖 5.10

5.5 SAKM 中的退化現象

在執行 SAKM 演算法時，若點數過少（ 如 20 點分 3 群 ）或群數過多（ 如 200 點分 32 群 ）或初始溫度過高時，常會使分群結果低於指定分群數目，最明顯的特徵是其 TSSE 對總執行次數曲線有回升，如圖 5.11。

造成分群結果退化的原因在於重分配時並未對各群數目做檢查，使得有些小群完全被改分配到它群而完全消失並使 TSSE 增加。

改善方式只要在重分配時並對各群數目做檢查，當該群點數少於某下限值時不改分配即可。

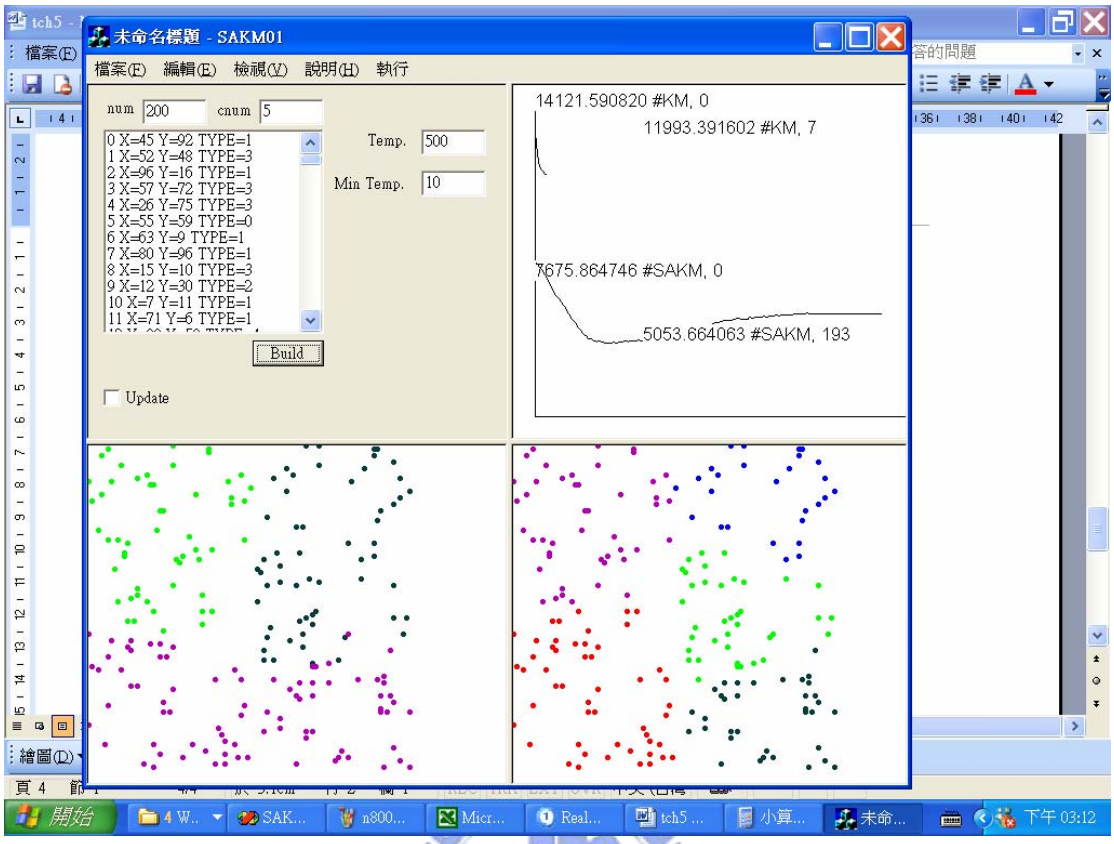
未命名標題 - SAKM01
檔案(F) 編輯(E) 檢視(V) 說明(H) 執行
num 200
cnum 5
0 X=45 Y=92 TYPE=1
1 X=52 Y=48 TYPE=3
2 X=96 Y=16 TYPE=1
3 X=57 Y=72 TYPE=3
4 X=26 Y=75 TYPE=3
5 X=55 Y=59 TYPE=0
6 X=63 Y=9 TYPE=1
7 X=80 Y=96 TYPE=1
8 X=15 Y=10 TYPE=3
9 X=12 Y=30 TYPE=2
10 X=7 Y=11 TYPE=1
11 X=71 Y=6 TYPE=1
Build
Temp. 500
Min Temp. 10
Update
14121.590820 #KM, 0
11993.391602 #KM, 7
7675.864746 #SAKM, 0
5053.664063 #SAKM, 193

圖 5.11

參考文獻

[1] U. Maulik and M. K. Pakhira, “Clustering using Simulated Annealing with Probabilistic Redistribution”, International Journal of Pattern Recognition and Artificial Intelligence, vol.15, no.2, pp.269-285, 2001.
[2] M. B. H. Rhouma and H. Frigui, “Self-Organization of Pulse-coupled Oscillators with Application to Clustering”, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol22, no.2, pp.180-195, 2001.
[3] H. Frigui and R. Krishnapuram, “Clustering by Competitive Agglomeration, Pattern Recognition”, vol.30, no. 7, pp. 1109-1119, 1997.
[4] L. Y. Tseng, S. B. Yang, “A Genetic Clustering Algorithm for Data with Non-Spherical-Shape Clusters”, Pattern Recognition, vol.33, pp.1251-1259, 2000.
[5] U. Maulik and S. Bandyopadhyay, “Genetic Algorithm-based Clustering Technique”, Pattern Recognition, vol.33, pp1455-1465, 2000.
[6] http://rx.mc.ntu.edu.tw/~jlin/Pharmacoinformatics/node29.html