遺傳演算法於資料倉儲中構建資料方體之應用

Configuring Data Cubes in Data Warehousing Environment with Genetic Algorithms

林文揚
Wen-Yang Lin
義守大學資訊管理學系
Dept. of Information Management
I-Shou University, Kaohsiung
wylin@csa500.isu.edu.tw

郭義中
I-Chung Guo
義守大學資訊工程研究所
Institute of Information Engineering
I-Shou University, Kaohsiung
m873205m@csa500.isu.edu.tw

摘要

資料倉儲是針對決策支援系統的需求所發展出的新一代資料庫的觀念，其資料通常經由線上分析處理，提供管理者決策時的參考。爲縮短查詢的時間，並提供使用者各個不同的觀察角度，這些資料通常在某一主題的關聯下，以多維度的資料型式儲存，稱爲資料方體。資料方體中的每一方格代表使用者所關心的，某一經合計後的視域。資料方體構建的問題即是，給定一主題及相關的維度所組成的資料方體，考慮使用者欲進行的查詢問題，探討在有限的儲存空間限制下，如何選取適當的子方體(視域)加以實體化，以縮短查詢的時間。這個問題已知是屬於NP完全問題。

本論文針對此問題，嘗試以遺傳演算法建立一子方體實體化選擇的模式，以建立資料倉儲中的資料方体。我們提出一貪婪式的遺傳演算法，稱爲遺傳貪婪法。根據實驗的結果，由遺傳貪婪演算法所找到的解遠優於過去廣爲採用的貪婪方法；在相同的儲存空間限制下，能大幅地減少處理查詢所需的成本。

關鍵字: 資料倉儲、資料方體、實體化視域、遺傳演算法、貪婪方法。

Abstract

Data warehousing is a new database concept dedicated to supporting executive managers in decision making through online analytical processing (OLAP). To decrease the query time and provide various viewpoints, these data usually are organized as a multiple dimensional data model, called data cubes. Each cell in a data cube represent one of the aggregation of manager's concern. The data cube configuration problem is, given a set of user queries and a storage constraint, to select a set of materialized subcubes from the data cubes to minimize the query cost, such as response time and/or the maintenance cost. This problem is known to be an NP-complete problem.

In this paper, we concern the application of genetic algorithms to this problem. We proposed a greedy-repaired genetic algorithm, called genetic greedy method. According to our experiments, the solution obtained by our genetic greedy method is superior to that found by traditional greedy method. That is, within the same storage constraint, the solution can greatly reduce the amount of query cost.

Keywords: data warehousing, data cubes, materialized views, genetic algorithm, greedy method

壹、緒論

資料倉儲是針對決策支援系統的需求所發展的新一代資料庫的觀念。其背景主要是許多企業長久累積了大量的交易資料，並冀望藉由分析這些交易資料以支援決策的工作。在分析的過程中，使用者往往需要從各個不同的角度觀察資料，並且快速的變換觀察的角度。爲了滿足此種分析的模式，資料倉儲將各個來源資料庫的資料，經過處理後加以儲存。這些資料通常以多維度(multidimensional)的資料型式儲存[3]，稱爲資料方體(data cube)。資料方體中的每一資料方格(cell)代表使用者所關心的，某一經聚合(aggregation)後的視域。由於資料倉儲的儲存空間有限，因此在建立這些資料方體時，就必須有所取捨，選擇較合適的資料方格加以儲存。資料方體構建的問題即是，給定一主題及相關的維度所組成的資料方體，考慮使用者欲進行的查詢問題(queries)，探討在有限的儲存空間限制下，如何選取適當的子方體(視域)加以實體化(materialize)，以縮短查詢的時間。這個問題已知是屬於*NP*完全問題[10]。

本篇論文的目的就是探討資料倉儲中，如何選擇適當的子方體做實體化的動作。不同於過去此問題的相關研究大多採取啓發式(heuristic)的方法，我們嘗試應用遺傳演算法(Genetic Algorithms)來解決此一最佳化的問題。我們發現，直接將遺傳演算法套用在資料方體構建的問題上，會面臨不合理解及搜尋速度緩慢的難題。爲了解決此現象，我們在染色體的修補過程中加入了貪婪法(greedy method)的觀念；經過實驗顯示，此一作法能有效的解決上述的問題。另外，我們也將我們的遺傳貪婪法與過去此研究所採用的貪婪法[1, 8, 10]做一比較，經實驗發現，遺傳貪婪法確能求得較佳的解。

本論文其餘的章節安排如下：第二節中，我們首先介紹資料倉儲的基本觀念與架構，然後詳細定義我們欲解決構建資料方體的問題，並說明過去的相關研究。在第三節中，首先說明遺傳演算法的觀念及架構，然後敘述如何將遺傳演算法應用在資料方體構建的問題上。第四節描述我們實驗的方式及實驗的結果，並與現行的貪婪法作比較。第五節爲此篇論文的結論。

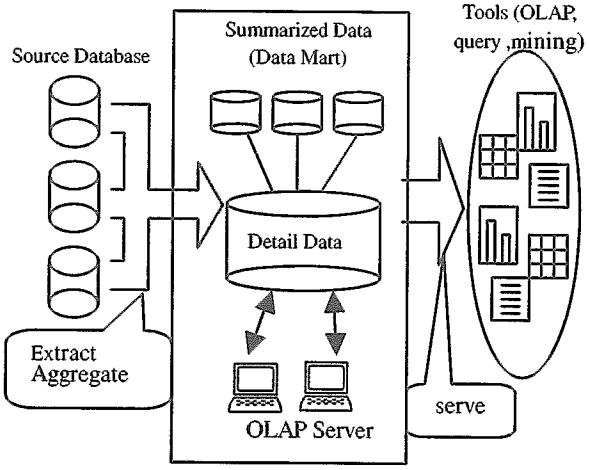
貳、問題背景描述

2.1 資料倉儲

資料倉儲是 Inmon[12]提出的新一代決策支援系統的觀念。其背景主要是許多企業長久累積了大量的資料，這些資料對企業的經營者而言是極重要的資產，除了作爲某些作業的記錄，以進行追蹤稽核之外，更是決策者進行決策分析時最主要且客觀的資料來源。然而這些資料常存在於不同的資訊系統中，其意謂著決策者進行決策分析時無一共通的資料來源，這導致至少下列幾個重要的缺失：1)決策分析者不知從何處獲得相關的資料；2)當不同的分析者引用不同的資料來源時，極可能產生不同的結論；3)由於資料分散且來源的資訊系統不同，分析者進行分析時需先進行資料的重整，包括格式轉換與篩選的工作，導致處理的時間過長，所得結果往往不具時效性。

爲解決上述問題，資料倉儲的技術在於根據分析者的需求，將各個來源資料庫的資料經過萃取(extract)及轉換(transform)等處理後儲存。當使用者執行任一查詢時，只要從資料倉儲處找尋相關資訊即可，無須從外部的資料庫來源尋找資料，如此便能大幅縮短查詢的時間。然而，因爲資料爲事先儲存，所以預設的查詢多爲一般常見或可預期的查詢。所以一旦使用者所下的查詢並不在當初預設的查詢中，很有可能面臨無法回答或資料不完整的可能。此外使用者也無法從資料倉儲中，獲得最新的資料。

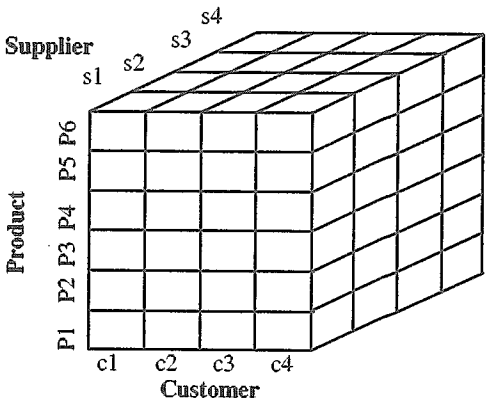
一完整的資料倉儲主要由三個部分所組成[3]，包括後端的資料來源(source databases)，核心部分的資料倉儲及資料超市(data marts)伺服，以及前端的分析工具，通常是線上分析處理(Online Analytical Processing, 簡稱 OLAP)、查詢/製表工具及資料挖掘(data mining)軟體，其典型的架構如圖一所示。



圖一、資料倉儲的架構。

2.2 資料方體與視域選取問題

由於資料倉儲的資料是來自各個來源資料庫，在相關主題(subject)關連下，經過萃取及轉換後儲存，因此資料倉儲可以視爲一堆實體化視域(materialized views)的儲藏所(repository)。如圖一所示，資料倉儲所支援的前端分析工具之一爲線上分析處理，此種分析的特點在於能提供使用者對所欲分析的資料，如產品的銷售額，從許多不同的角度，如產品、顧客、供應商等，進行所謂的聚合查詢。例如以銷售資料爲例，分析者關心的話題多半環繞在消費者(customer)、供應商(supplier)、零件(part)、及銷售額(sales)。所以資料倉儲中有關銷售額的資料可以視爲一個包含消費者、供應商及零件三個維度的立方體，即資料方體。每一個立方體中的資料格，代表銷售額在這三個維度上的表現，即某一家供應商所供應、賣給某一客戶的某一零件的銷售額，如圖二所示。

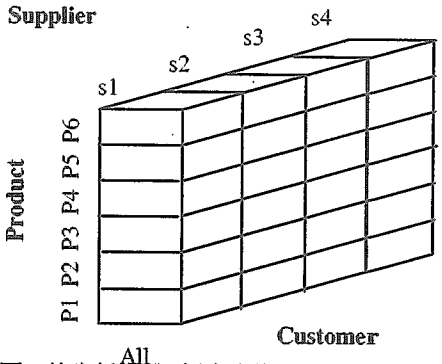


圖二、資料方體的例子。

從圖二的資料方體中，我們可以看出有 4 位供應商、4 個客戶以及 6 項產品。如果使用者欲查詢每個供應商供應各個零件的銷售情形，那麼我們必須對所有的顧客做加總的動作，例如(c1, p1, s1) + (c2, p1, s1) + (c3, p1, s1) + (c4, p1, s1)，表示供應商 s1 所供應的產品 p1 的銷售額，其餘供應商與產品的銷售情形可依此方式求出。上述的查詢可以下列的 SQL(Standard Query Language)語言表示:

```
SELECT Product, Supplier, SUM (Sales) AS Total Sales
FROM R
GROUP BY Product, Supplier;
```

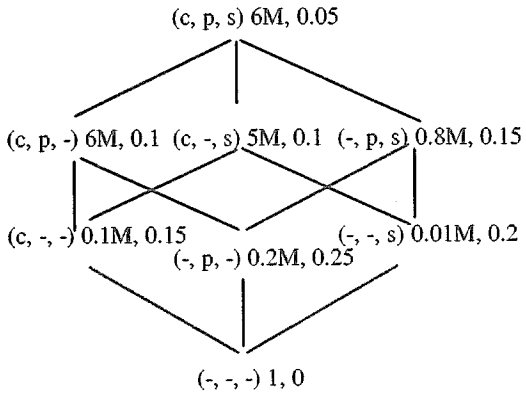
其中 R 是我們資料的來源。此查詢的結果實際上是對應原來的資料方體中的某一子方體(subcube)，如圖三所示。



圖三、圖二的資料方體，經查詢後的結果對應的子方體。

依這樣的結果，我們可以將上述的查詢所對應的子方體以(-, p, s)的形式表示，符號"-"表示其所對應的屬性(維度)不在 SQL 的 GROUP BY 子句中。因此，由消費者(customer)、供應商(supplier)及零件(part)這三個維度所組成的資料方體，當我們對任意維度作加總後，實際上可以衍生出八種可能的子方體(即視域)。圖四顯示此八個子方體間的關係可以形成一方格(lattice)，子方體後的數字分別表示其大小及使用的頻率。其中兩個子方體 D_1、D_2 間的連線表示此兩子方體間的依賴關係，記成 $D_1 \pi D_2$，表示組成 D_1 的屬性包含於 D_2，即 $A(D_1) \subset A(D_2)$。其意義爲，凡是可以由 D_1 回答的查詢也可以由

D_2所回答，反之則不然。例如，考慮(-, p, -) ≤ (-, p, s)。若使用者想了解所有的產品的銷售狀況，很顯然的我們可以由 (-, p, -) 或 (-, p, s) 子方體來回答。但是若使用者想進一步了解各供應商所供應的產品的銷售狀況就只能由 (-, p, s) 子方體來回答。



圖四、一由八個子方體組成的方格。

由上面的討論可以得知，若要加快處理查詢的速度，資料倉儲就必須針對使用者會詢問的問題，從資料方體中挑選合適的子方體加以儲存，稱之爲實體化。最好的方式當然是將所有的子方體都實體化。然而，資料倉儲的實際空間，通常無法儲存所有的子方體，因此，必須就使用者所查詢的問題，在有限空間的考量下，挑選最佳的子方體集合加以實體化，使回答所有的查詢時間爲最短。此一最佳化的問題就稱之爲實體化子方體或實體化視域的選取問題。這類的問題，已知是無法用多項式時間內所可以解決的問題，也就是 *NP* 完全的問題。

2.3 相關研究

在資料庫研究的領域中，許多學者早已發覺實體化視域能縮短查詢的時間[4, 14, 15]。這幾年隨著資料倉儲的發展，逐漸有學者探討資料倉儲中視域實體化的選取問題。綜合這些學者的研究，我們略敘如下：Harinarayan 等人[10]首先定義在處理 OLAP 的多維度分析時，如何選取實體化的子方體的問題，並提出一貪婪演算法來解決此問題；[8]則進一步將索引(index)的選取納入考量，但二者都未考量維護這些實體視域的成本[18, 19]。Baralis 等人[2]將 Harinarayan 等人所定義的資料方體的方格模式(lattice model)進一步正規化，並討論如何根據使用者的查詢，找出方格模式中相關的子方體，以減少可能解的空間(solution space)。Ross 等人[19]則討論如何實體化一些額外的視域，以減少視域維護的成本。Gupta[7]針對視域實體化的選取問題作了有系統的討論；其考慮的查詢種類擴大爲可表示成 AND-OR 圖的查詢，在有限的儲存空間下，如何選取實體視域或其索引，以減少查詢處理及視域維護的成本；最近則又探討在視域維護成本的限制下，如何選擇實體化的視域[9]。在[21]，Theodorates 等人亦探討了此一問題，其研究進一步假設所選取的實體化視域必須能回答所有的查詢問題，或使用者的查詢經改寫(rewritting)後，仍然可用選取的實體化視域來回答，但去掉了儲存空間的限制。在同一時間，Yang 等人[22]也針對同一問題提出不同的處理模式，其考慮多個查詢間通常存在共同的部分表達式(common subexpressions)的現象[13]，而在 1999 年 Yang 等人[23]亦提出利用遺傳演算法解決視域選取的問題，並證明此方法優於某些啓發式的演算法。至於國內的研究部分，則有陳耀輝等[1]針對此問題提出一兩階段的演算法，先減少資料倉儲的儲存空間，再改進 Harinarayan 等所提的貪婪法來挑選合適的視域。

由於視域實體化的選取問題是屬於 NP 完成問題，目前解決此一問題主要有兩種方法：一種是窮舉法(exhaustive method)[21]或加上某些修剪(pruning)的方法；另一種則以啓發式(heuristic)爲主[1, 7, 8, 10, 22]，如貪婪法。前者的問題在於複雜度太高，而後者所得到的解很難驗證其優越性。因此，如何兼顧時間複雜度與所得解的品質仍是亟待研究的課題。

參、應用遺傳演算法於子方體的選擇

3.1 遺傳演算法

遺傳演算法是 John Holland 於 1975 年[11]提出來，以演繹的方式求解的方法。它將達爾文的物競天擇觀念加入，根據優勝劣敗的原則，挑選適合生存的個體。然後利用生物學上的交配與突變的動作來產生後代。經過這樣不斷的替換動作，產生出來的每一個子代，理論上會比上一代好，最終可以獲得一個包含近似最佳解的個體。遺傳演算法的優點在於透過物競天擇、適者生存的取代策略及基因運算，可以避免在搜尋時陷入區域性的最佳解的陷阱，而且可以減少許多尋找最佳值過程所需的時間，是故非常適用於最佳化的問題[7, 17, 18]。下列是一個簡單的遺傳演算法[7]。

演算法一：簡單的遺傳演算法(Simple Genetic Algorithm)。

步驟 0: 定義個體的表示方式，族群數目，終止演繹的條件，評估函數，交配方法，突變方法，交配率，突變率等。

步驟 1: 產生含有 N 個個體 $x_1, x_2, \ldots, x_N$ 的起始族群 P。

步驟 2: 使用評估函數計算此族群中每個個體的適應值(fitness)。

步驟 3: 當終止演繹的條件還未成立前，執行步驟 4~6。

步驟 4: 重複下列步驟直到產生包含 N 個個體的新族群 P'：

1) 由目前的族群中選擇兩個個體作爲父染色體，個體被選取到的機率通常設爲其適應值與整個族群的適應值總合的比率。
2) 根據交配率，將父染色體進行交配產生兩個子染色體；若無須進行交配，則直接將父染色體內容複製到子染色體。
3) 根據突變率，將子染色體進行突變後置入新的族群。

步驟 5: 以新的族群 P' 取代目前的族群 P，$P \leftarrow P'$。

步驟 6: 跳回步驟 2。

通常的終止條件可設爲

1. 達到預設的代數。
2. 每一代的適應值已經收斂。

3.2 運用傳統的遺傳演算法

我們首先討論如何運用上述的遺傳演算法來實作子方體的選擇問題。根據演算法一的架構，我們必須決定組成遺傳演算法的各個機制或參數，即步驟 0 的工作，分述如下：

編碼方式

即個體或染色體的表示方式，其代表如何將問題的可能解表示成一基因(gene)串列，這是運用遺傳演算法解決問題時，最基本的前置工作。一般最常見的編碼方式，是將問題的可能解化成一串由二進位所組成的字串。就子方體實體化的選擇問題來說，此爲一極自然的編碼方式。例如，以圖三的八個子方體爲例，我們可以將某一選取的方式以長度爲 8 的 0, 1 的字串表示，0 的位置即表示該子方體不予選取，1 則表示選取，故 01011010 即表示所選取的子方體集合爲 (c, p, -)、(-, p, s)、(c, -, -)、(-, -, s)。

適應函數

每一個體在演化的過程中，都會被賦予一個數值，以表示此個體在族群中的優秀程度，稱爲適應值；而用以計算個體適應值的函數，就稱之爲適應函數。適應函數的制定對遺傳演算法的成敗具有決定性的影響，必須針對問題本身的特性來設計，不同的問題有不同的定義方式，在遺傳演算法的各機制中，可說是最不具通用性的部份。讓我們再來回顧我們所探討的子方體實體化問題。假設所考慮的資料方體的所有可能的子方體爲 $D_1, D_2, \ldots, D_n$，使用者所查詢的聚合問題爲 $Q_1, Q_2, \ldots, Q_m$，每個問題查詢的頻率爲 $f_{Q_1}, f_{Q_2}, \ldots, f_{Q_m}$，回答查詢 Q_i 所需的查詢成本爲 $C(Q_i)$，而資料倉儲可以儲存的空間爲 S，則子方體實體化問題可以表示成：求取一組向量 $V = \langle v_1, v_2, \ldots, v_n \rangle$，其中 $v_i \in \{0, 1\}$，使得

$$\sum_{i=1}^{n} v_i \cdot |D_i| \le S \ , \qquad (1.1)$$

且 $\sum_{i=1}^{m} f_Q \cdot C(Q_i)$ 爲最小。 (1.2)

上述的式子的求解最主要的關鍵，在於如何計算回答查詢的成本。根據 Harinarayan 等人的實驗[10]，回答聚合查詢如加總、平均等類問題，所需的時間與視域的資料筆數成正比。因此，我們所需要知道的是每個查詢需用何種子方體來回答。此對應的關係可以根據查詢中所牽涉的屬性與組成子方體的維度作一比對即可。舉例來說，考慮前述產品、顧客、供應商的例子，若使用者欲了解各產品的銷售額，則直接對應的子方體爲(c, -, -)，所需的成本爲 0.1M。然而，若是(c, -, -)未實體化，則根據子方體間的相依關係(如圖四)，可以使用其所依賴的子方體中，資料筆數最少，且有實體化者來回答此查詢，如(c, -, s)，則所需的成本變爲 6M。

由上述的討論可以進一步發現，回答所有問題的查詢時間，實際上等於子方體被使用的時間總合。因此，上述的問題可以簡化爲

$$\sum_{i=1}^{n} v_i \cdot |D_i| \le S \ , \qquad (2.1)$$

且 $\sum_{i=1}^{m} f_D \cdot C_V(D_i)$ 爲最小。 (2.2)

此處 $f_{D_1}, f_{D_2}, \ldots, f_{D_n}$ 表每個子方體被使用的頻率，而 $C_V(D_i)$ 表當子方體實體化的向量爲 V 時，使用子方體 D_i 的成本爲 $C_V(D_i)$。

挑選方式

挑選指的是如何從群組中選取個體，進行遺傳運算如交配，以產生下一代。根據優勝劣敗的觀念，通常是儘量挑選較優的個體以產生較佳的下一代。然而，若是都集中在少數較優秀的個體，則會使族群的變異性(diversity)降低，意即可能解的搜尋範圍縮小，導致整個族群很快就收斂，無法繼續演繹，求取更佳的解。在此我們採用競賽選取法(tournament selection)[17]，其方法爲自目前的族群中，隨機地選取兩個個體，並隨機產生一介於 0 至 1 間的實數 r，若 $r < k$ (k 爲一事先設定的參數，通常設爲 0.075)，則挑選兩者中適應值較高的個體，否則挑選適應值較低的個體；被選中的個體下次仍有機會可以再被挑選。另外，根據 K. De Jong [17]的研究，將上一代族群中最好的個體保留至下一代的族群中，一方面能加快收斂的速度，又能求得較佳的解。故此，我們也在我們的方法中加入了此作法。

交配運算

遺傳演算法產生下一代的方法，就是透過交配，選定兩個父母代，做交配的動作，產生兩個子代。最常用的是單點交配，也就是在兩個個體中，隨機選取一定點，將染色體切成一半，做字串交換的動作。如此新的個體就具有上一代的部分特色，但又不會與上一代完全相同。

突變運算

如同生物的演化一樣，染色體也有偶發的突變，使演化多點變化性，其作用在提高族群的變異性，避免陷入區域的最佳解。最常用的突變是針對染色體的某一位元，做 0 與 1 互換的動作，將原本數字爲 0 的位元換成 1，原本數字爲 1 的換成 0。由於突變率通常很低，所以實際運作時，突變動作的次數不多。

3.3 遺傳貪婪演算法

前述的簡單遺傳演算法，經過一連串的運算，會產生不合理的解，其原因主要是編碼的方式並未考慮問題本身的條件限制。以圖四的例子爲例，假設空間的限制爲 7M，在進行交配運算時所挑選的個體分別爲 10001101 及 00111110，很顯然的這兩個個體所對應的子方體集合，所需的空間都符合 7 M 的限制。然而，令所取的交配點爲第 2 個基因，則所產生的兩個子代爲 10111110 及 00001101，很顯然地第一個子代無法滿足空間的限制，即不合理的解。要解決不合理的解的問題有許多的作法[16]，常見有三種：一種是改變個體的編碼方式，避免產生不合理的解；另一種則是使用懲罰函數(penalty function)，在適應函數之後，減去一所定義的懲罰函數，儘可能地讓不合理的解得到極差的適應值，以減少其存留的機會；第三種則是使用修補的方法(repair method)，將個體的基因作調整，以符合問題的條件限制。根據過去學者的相關研究顯示[16]，使用修補的方法通常有較佳的表現，因此，我們採取第三種作法。

所謂修補方式通常是把錯誤的染色體，一次修改一個基因的值，直到符合我們的限制條件爲止。以子方體實體化的問題爲例，即是一次挑選一個基因值爲 1 的基因，將其改成 0，直到此個體所代表的子方體集合的空間小於或等於儲存空間的限制。此種修補方式會直接影響到所求得解的優劣，而其關鍵就在於基因挑選的方式。

最簡單的方式是由染色體中，隨機挑選任一位元值爲 1

的基因，做修補的動作，修正完後，再隨意挑選下一個。不過這種修補方式，不容易達到最佳的修正。另一種作法則是先將所有已挑選到的子方體，根據其大小來排列，然後每次挑選目前最大或最小的子方體。此種類似貪婪法的作法的缺點，在於無法反應被挑選的子方體的價值，即此子方體被捨去後，會增加整個查詢多少的時間。在此，我們針對子方體實體化問題的特性，提出另一貪婪式的修補方法，詳述如下。

我們首先定義一計算每個子方體的痛苦指數(Paining function)，$P_V(D_i)$如下：

$$P_V(D_i) = G_V(D_i) / |D_i| \quad (3.1)$$

$$G_V(D_i) = \sum_{i=1}^{n} f_{D_i} \cdot \left(C_{V>i}(D_i) - C_V(D_i)\right) \quad (3.2)$$

上述 $P_V(D_i)$代表的意義爲，當目前的子方體實體化向量爲 V 時，將子方體 D_i 移除後，回答所有的查詢所增加的時間 $G_V(D_i)$與 D_i 大小的比值，即單位空間所增加的查詢時間，而符號 $V>i$ 即表示將子方體 D_i 移除後所對應的實體化向量。根據此定義，計算出在實體化向量 V 中，所有已挑選到的子方體的痛苦指數 $P_V(D_i)$，其指數最低的子方體所對應的基因，即爲修補的對象。我們的貪婪修補方法如下：

演算法二、貪婪修補法(Greedy repaired method)。

步驟 0: 令欲進行修補的個體爲 x，且 $V \leftarrow x$。

步驟 1: 當 $\sum_{i=1}^{n} v_i \cdot |D_i| > S$ 時，重覆執行步驟 2 ~ 5。

步驟 2: 根據式子(3.1)及(3.2)計算所有 $v_i = 1$，$1 \le i \le n$，的子方體 D_i的痛苦指數 $P_V(D_i)$。

步驟 3: 從步驟 1 的結果中，挑選痛苦指數最小的子方體，設爲 D_j。

步驟 4: $V \leftarrow V>j$。

步驟 5: 跳回步驟 1。

舉例來說，讓我們考慮前述的例子：需進行修補的個體爲 10111110，空間限制爲 7M，且假設這八個子方體被使用的頻率分別爲 0.05, 0.1, 0.1, 0.15, 0.15, 0.25, 0.2, 0，如圖所示。另外我們假設，當(c, p, s)未實體化時，所有須使用(c, p, s)來回答的查詢必須回到資料倉儲中的基底關聯資料(base relation)，在此我們設此基底關聯資料筆數爲 200M。根據式子(3.1)及(3.2)，可求得(c, p, s)、(c, -, s)、(-, p, s)、(c, -, -)、(-, p, -)及(-, -, s)等六個子方體的痛苦指數，如表一所示。其中任一子方體 D_i 的"受依賴的子方體"一欄中的子方體，表示這些子方體所對應的使用者查詢，可用以回答的實體化子方體中的最小者即爲 D_i。例如，以(c, p, s)爲例，在執行第一次挑選時，須使用此子方體來回答使用者查詢的子方體除其本身外，還有(c, p, -)，因爲(c, p, -)本身並未實體化。因此，當(c, p, s)被移除後，原本可由其回答的查詢就必須回到基底關聯，而增加的時間即是(c, p, s)及(c, p, -)所對應的查詢，依此可求出其痛苦指數爲(200- 6)* (0.05+0.1)/6= 4.85。由表一的結果得知，最後所挑選出須移除的子方體爲(c, -, s)及(-, p, -)，而剩餘的實體化子方體的空間爲 6.91M，小於 7M 的空間限制。

表一、在執行貪婪修補過程中，所挑選的子方體的痛苦指數。

	第一次挑選		第二次挑選	
	受依賴的子方體	痛苦指數	受依賴的子方體	痛苦指數
cps	Cps, cp-	(200-6)*0.15/6=4.85	cps, cp-, c-s	(200-6)*0.25/6=8.08
c-s	c-s	(6-5)*0.1/5=**0.02**	c-s	
-ps	-ps	(6-0.8)*0.15/0.8=0.975	-ps	0.975
c--	c--	(5-0.1)*0.25/0.1=12.25	c--	12.25
-p-	-p-	(0.8-0.2)*0.25/0.2=0.75	-p-	**0.75**
--s	--s	(0.8-0.01)*0.2/0.01=15.8	--s	15.8

整個遺傳貪婪演算法如下：

演算法三：遺傳貪婪演算法(Genetic Greedy Algorithm)。

步驟 0: 定義個體的表示方式，族群數目，終止演繹的條件，評估函數，交配方法，突變方法，交配率，突變率等。

步驟 1: 產生含有 N 個個體 $x_1, x_2, \ldots, x_N$ 的起始族群 P；若任一個體爲不合理解，使用貪婪修補法改正。

步驟 2: 使用貪婪修補法改正族群中不合理的個體，然後使用評估函數計算此族群中每個個體的適應值(fitness)。

步驟 3: 當終止演繹的條件還未成立前，執行步驟 4~6。

步驟 4: 重複下列步驟直到產生包含 N 個個體的新族群 P'：

1) 由目前的族群中選擇兩個個體作爲父染色體，個體被選取到的機率通常設爲其適應值與整個族群的適應值總合的比率。
2) 根據交配率，將父染色體進行交配產生兩個子染色體；若無須進行交配，則直接將父染色體內容複製到子染色體。
3) 根據突變率，將子染色體進行突變後置入新的族群。

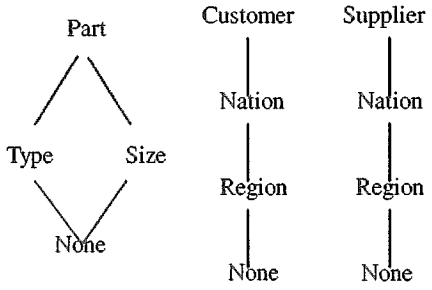
步驟 5: 以新的族群 P'取代目前的族群 P，$P \leftarrow P'$。

步驟 6: 跳回步驟 2。

肆、實驗與分析

在本節中，我們將說明我們所進行的實驗與結果的分析。我們用以測試的資料庫，是 TPC-D 決策支援測試庫[20]，這個資料庫是根據交易處理效能會議(Transaction Processing Performance Council，簡稱 TPC)，在 1995 年針對決策支援所制定的測試資料庫。這個資料庫最多可以產生高達 10000GB 的資料，我們所用的是其中的最小的測試庫，1GB。我們從這資料庫中挑選三個維度來組成測試用的資料方體，分別爲顧客(Customer, c)、零件(Part, p)、及供應商(Supplier, s)等，而爲了增加子方體的組合，我們又在每個維度中，挑選部份的屬性形成階級(hierarchy)，如圖五。由於零件的部分組合有四種，顧客有四種，同樣供應商也是四種，所以所有的子方體組合共有 64 種。

茲將所有的子方體及其大小列述於表二，其中子方體的屬性都以第一個英文字母來表示，"-" 的符號爲 none 的意思，空間單位爲 K(1000)資料筆數。至於各個子方體的使用頻率，我們考慮了三種組合：第一種是所有的子方體的使用頻率皆相同，且都大於零；第二種是



圖五、TPCD 中，Part、Customer、及 Supplier 這三個維度的屬性的階層關係圖

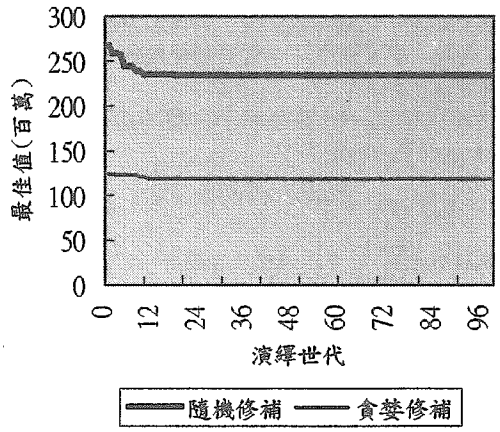
由隨機產生介於 0~1 間的亂數所組成；第三種則是先將所有子方體由小至大排序，然後將其頻率設成子方體的大小的等第(ranking)的倒數；換言之，愈大的子方體其頻率將愈小。

表二：TPCD 中，由 customer、part、supplier 所組成的所有子方體。

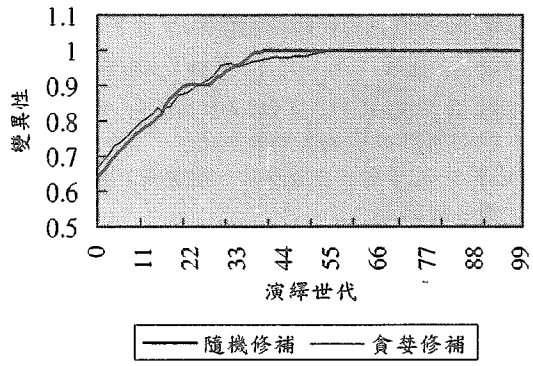
cps	6000	nps	5000	rps	4000	-ps	800
cpn	6000	npn	5000	rpn	4000	-pn	800
cpr	6000	npr	5000	rpr	4000	-pr	800
cp-	6000	np-	5000	rp-	1000	-p-	200
css	5000	nss	500	rss	2500	-ss	500
csn	5000	nsn	30	rsn	6.25	-sn	1.25
csr	5000	nsr	6.25	rsr	1.25	-sr	0.25
cs-	5000	ns-	1.25	rs-	0.025	-s-	0.05
cts	5990	nts	800	rts	3000	-ts	1500
ctn	5990	ntn	90	rtn	18.75	-tn	3.75
ctr	5990	ntr	18.75	rtr	3.75	-tr	0.75
ct-	5990	nt-	3.75	rt-	0.75	-t-	0.15
c-s	6000	n-s	250	r-s	50	--s	10
c-n	2500	n-n	0.625	r-n	0.125	--n	0.025
c-r	500	n-r	0.125	r-r	0.025	--r	0.025
c--	100	n--	0.025	r--	0.005	---	0.001

另外我們考慮 9 種儲存空間的限制，分別爲子方體總合的 5%、10%、20%、30%、40%、50%、60%、70% 及 80%。如此，配合使用頻率，形成 27 組不同的測試組合。此外，若發生當某一查詢所使用的子方體皆未實體化的情況，我們則將其視爲需回到資料倉儲的基底關聯進行查詢，在此我們將其時間設爲資料方體中最大的子方體的 3 倍，即 18M。

我們首先進行的實驗是比較幾種修補方法的效能。在此我們比較前述的隨機修補及我們的貪婪修補方法，所使用的的測試組合爲頻率相同、儲存空間 20%；實驗的結果如圖七及圖八。

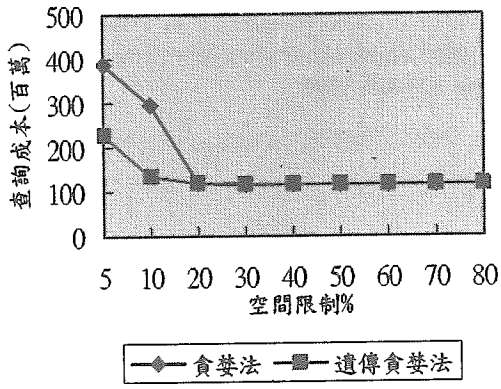


圖七、隨機修補與貪婪修補作法的遺傳 演算法，在求得的解的適應值的比較

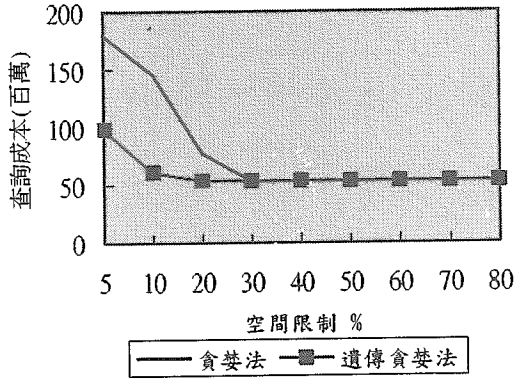


圖八、隨機修補與貪婪修補作法的遺傳演算法，在族群收斂情形的比較。

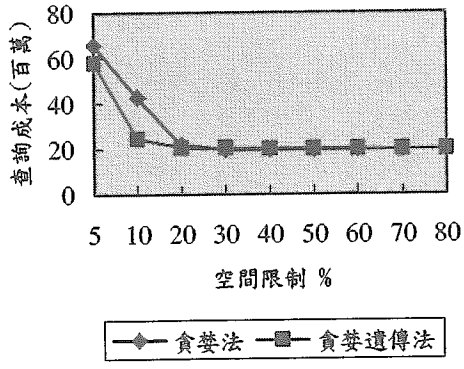
由結果可以看出，貪婪作法的修補方式確實能大幅提高所求得解的品質，在此例子大約爲 2 倍；而觀察族群中的個體差異的程度，亦可看出貪婪的修補方式，能延緩族群的整體收斂，以保持較大的變異性，求得更佳的解。接著我們比較所構想的遺傳貪婪法與傳統的貪婪法，以測試遺傳貪婪法的效能。我們根據子方體使用頻率的特性，分成三組進行測試，結果如圖九至十一。



圖九、當各子方體的使用頻率皆相同時， 貪婪法與遺傳貪婪法的比較。



圖十、當各子方體的使用頻率爲隨機亂數時，貪婪法與遺傳貪婪法的比較。



圖十一、各子方體的使用頻率與其大小成反比時，貪婪法與遺傳貪婪法的比較。

由這些結果，我們觀察到幾個重要的現象：

1) 無論所測試的資料方體可儲存的空間與使用頻率的組合爲何，我們的遺傳貪婪法皆能求得比傳統貪婪法更佳的解，特別是當儲存的空間小於 20%時，其差距尤其明顯。

2) 資料倉儲中用來儲存實體化子方體的空間，不需要超過整個子方體的總合的 20%，因爲由實驗結果可看出，當空間繼續增加時，並無法進一步減少處理查詢所需的成本，此點與 Harinarayan 等人的研究結果[10]相符。

由上述的觀察，我們發現對子方體實體化選取的問題，空間的限制佔有極重要的地位—盲目地增加儲存的空間並不一定就能有效地減少處理查詢的時間。由我們的實驗可以發現，實際上最佳的儲存空間只需儲存全部可能的子方體的 20%，對不同的資料此比例會不同。因此我們認爲，在實際解決實體化子方體的問題上，應先估計該問題所需的最佳儲存空間。當實際可儲存的空間大於此空間時，空間限制只需要設成最佳儲存空間，採用傳統的貪婪法即可；但當儲存空間小於此空間時，應採用我們的遺傳貪婪法。特別當資料倉儲中的資料極爲龐大，且將來會繼續不斷地增加時，系統中可用以儲存實體化子方體的空間將遠小於所有子方體的總合，而就此點看來，我們所提出的遺傳貪婪法將是最佳的選擇。

伍、結論

在此篇論文中，我們針對資料倉儲構建過程中，支援多維度分析的子方體建立的選擇問題，探討使用遺傳演算法建立選擇模式的可行性。我們提出一植基於貪婪修補方式的遺傳演算方法，稱之爲遺傳貪婪法。根據實驗的結果，由遺傳貪婪法所找到的解遠優於廣爲採用的貪婪方法。而在實驗的過程中，我們亦發現儲存空間的重要性—儲存的空間有一最佳點，持續地增加儲存的空間並不一定就能有效地減少處理查詢的時間。在此一原則考量下，我們認爲在實作上應先估計該問題所需的最佳儲存空間。當實際可儲存的空間大於此空間時，空間限制只需要設成最佳儲存空間，採用傳統的貪婪法即可；但當儲存空間小於此空間時，則應採用我們的遺傳貪婪法求得最佳的子方體集合。至於如何有效地求得該問題的最佳儲存空間點，將是我們未來研究的重點。

參考文獻

[1] 陳耀輝、劉宇昌與劉佳灝，"在資料倉儲中選擇實體化視域之研究"，八十六年全國計算機會議論文集，第一輯，第 72-77 頁，1997。

[2] E. Baralis, S. Paraboschi, E. Teniente, "Materialized view selection in a multidimensional database," in *Proceedings of the 23rd VLDB Conference*, pp. 156-165, 1997.

[3] S. Chaudhuri and U. Dayal, "An overview of data warehouse and OLAP technology," in *Proceedings of ACM SIGMOD International Conference on Management of Data*, Vol. 26, pp. 65-74, 1997.

[4] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, and K. Shim, "Optimizing queries with materialized views," in *Proceedings of International Conference on Data Engineering*, pp. 190-200, 1995.

[5] C.I. Ezeife, "A uniform approach for selecting views and indexes in a data warehouse," in *Proceedings of International Database Engineering and Applications Symposium*, pp. 151-160, 1997.

[6] D.E. Goldberg, *Genetic Algorithms in Search,*

Optimization, and Machine Learning, Reading, MA, Addison-Wesley, 1989.

[7] H. Gupta, "Selection of views to materialize in a data warehouse," in *Proceedings of International Conference on Database Theory*, pp. 98-112, 1997.

[8] H. Gupta,V. Harinarayan, A. Rajaraman, and J.D. Ullman, "Index Selection for OLAP," in *Proceedings of International Conference on Data Engineering*, pp. 208-219. 1997.

[9] H. Gupta and I.S. Mumick, "Selection of views to materialize under a maintenance cost constraint," *International Conference on Database Theory*, Jerusalam, Israel, 1999.

[10] V. Harinarayan, A. Rajaraman, and J.D. Ullman, "Implementing data cubes efficiently," in *Proceedings of ACM SIGMOD International Conference on Management of Data*, pp. 205-216, 1996.

[11] J.H. Holland, *Adaptation in Natural and Artificial Systems*, Second edition, MIT Press, 1992.

[12] W.H. Inmon and C. Kelley, *Rdb/VMS: Developing the Data Warehouse*, QED Publishing Group, Boston, Massachussetts, 1993.

[13] M. Jarke, "Common subexpression isolation in multiple query optimization," *Query Processing in Database Systems*, pp. 191-205, 1984.

[14] Levy, A.O. Mendelson, Y. Sagiv, and D. Srivastava, "Answering queries using views," in *Proceedings of ACM Symposium on Principles of Database Systems*, pp. 95-104, 1995.

[15] P.-A. Larson and H.Yang, "Computing queries from derived relations," in *Proceedings of 1st VLDB Conference*, pp. 259-269, 1985.

[16] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, New York, 1994.

[17] M. Mitchell, *An Introduction to Genetic Algorithms*, MIT press, 1996.

[18] D. Quass, A. Gupta, I.S. Mumick, and J. Widom, "Making views self maintainable for data warehousing," in *Proceedings of International Conference on Parallel and Distributed Information Systems*, pp. 158-169, 1996.

[19] K.A. Ross, D. Srivastava, and S. Sudarshan, "Materialized view maintenance and integrity constraint checking : trading space for time," in *Proceedings of ACM SIGMOD International Conference on Management of Data*, pp. 447-458, 1996.

[20] F. Raab, ed., *TPC Benchmark™ D (Decision Support), Proposed revision 1.0*, Transaction Processing Performance Council, San Jose, CA, 1995.

[21] D. Theodoratos and T. Sellis, "Data warehouse configuration," in *Proceedings of the 23rd VLDB Conference*, Athens, Greece, pp. 126-135, 1997.

[22] J. Yang, K. Karlapalem, and Q. Li, "Algorithm for materialized view design in data warehousing environment," in *Proceedings of the 23rd VLDB Conference*, Athens, Greece, pp. 136-145, 1997.

[23] C. Zhang, X. Yao, J. Yang, "Evolving Materialized Views in Data Warehouse", in Proceedings of the 1999 Congress on Evolutionary Computations, Vol. 2, pp. 823-829, 1999.