啟發式資料方體挑選方法之分析比較

A Comparative Analysis of Heuristic Data Cube Selection Methods

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@isu.edu.tw

張耀升
Yao-Sheng Chang
義守大學資訊工程研究所
Institute of Information Engineering
I-Shou University, Kaohsiung
m893335m@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。為縮短查詢的時間，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，稱為資料方體。資料方體選取的問題即是，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非多項式時間完成問題。因此，目前已知的資料方體實體化的選取方法大多屬於啟發式的方法。本論文旨在比較這些啟發式的挑選方法，分析其效率及求解的品質，以了解這些方法的優劣及適用性。

關鍵詞: 資料方體、資料倉儲、啟發式方法、線上分析處理。

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. The data cube selection problem is, given the set of user queries and a storage space constraint, to select a set of materialized sub cubes from the data cubes to minimize the query cost, such as response time and/or the maintenance cost. This problem is known to be a NP-complete problem. Most of the existing algorithms are based on the greedy paradigm. In this paper, we compare and analyze the performance and quality of these greedy selection methods to rank their superiority and suitability.

Keywords: data cubes, data warehousing, heuristic method, OLAP.

一. 概論

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，並企望藉由分析這些大量的交易資料支援決策的工作。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。為了滿足此分析的需求，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，稱為資料方體(data cube)。資料方體中的每一資料格(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。資料方體建立的問題即是，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，如何選取適當的資料格(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非多項式時間完成問題(NP problem) [11]。目前解決此一問題主要有三種方法：一種是窮舉法(exhaustive method)[4, 23]，第二種為利用一些最佳化方法，如遺傳演算法[1]，第三種則以啟發式(heuristic)為主[2, 3, 7, 8, 10, 11, 21, 24]。

本篇論文的目的旨在比較現存的啟發式資料方體選取方法，分析這些方法的時間複雜度、執行效率與解的品質，以了解這些方法的優劣與適用時機，作為資料倉儲系統設計者的指引。

本論文其餘的章節安排如下：第二節中，首先介紹資料倉儲的基本觀念與架構，然後定義資料方體實體化的選取問題，並說明過去的相關研究。在第三節中，我們針對過去的研究所提出的各種貪婪挑選方法作一完整的說明。在第四節中，我們從理論上分析各方法的時間複雜度及各方法求得的解的效能。第五節描述實驗的方式並分析其結果。第五節為此篇論文的結論。

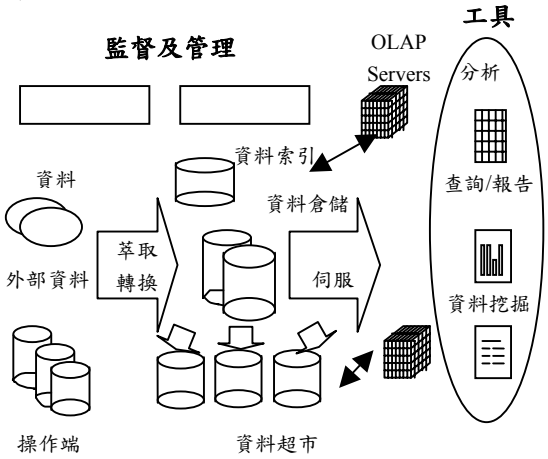
二. 問題背景描述

(一) 資料倉儲

資料倉儲是W.H. Inmon[13]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，除了作為某些作業的記錄，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。

為解決上述問題，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，因為資料為事先儲存，所以預設的查詢多為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，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(OLAP)、查詢/製表工具及資料挖掘(data mining)軟體[5]。其典型的架構如圖一所示。



圖一、資料倉儲的架構[5]。

(二) 資料方體選取問題

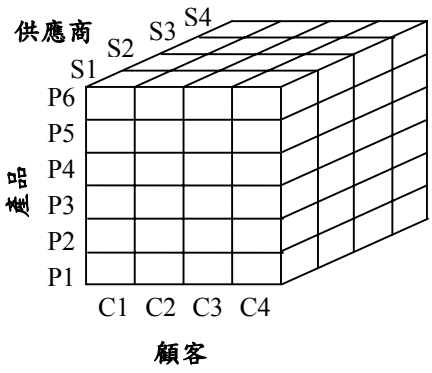
由於資料倉儲的資料是來自各個來源資料庫，在相關主題(subject)關連下，經過萃取及轉換後儲存，因此資料倉儲可以視為一堆實體化視域(materialized views)的儲藏所(repository)。如圖一所示，資料倉儲所支援的前端分析工具之一為線上分析處理，此種分析的特點在於能提供使用者對所欲分析的資料，如產品的銷售額，從許多不同的角度，如產品、顧客、供應商等，進行所謂的聚合查詢。例如以銷售資料為例，分析者關心的話題多半環繞在消費者(customer)、供應商(supplier)、零件(part)、及銷售額(sales)。所以資料倉儲中有關銷售額的資料可以視為一個包含消費者、供應商及零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。

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，也就是(c1, p1, s1) + (c2, p1, s1) +

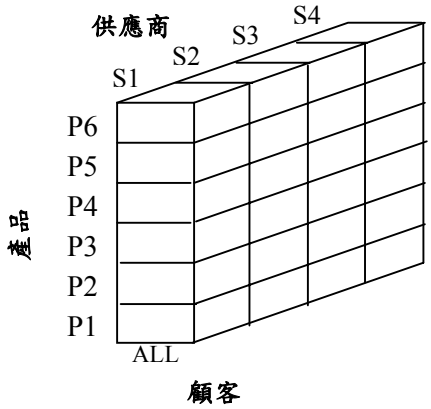
(c3, p1, s1) + (c4, p1, s1)。此表示供應商 s1 所供應的產品 p1 的銷售額，其餘供應商與產品的銷售情形可依此方式求出。上述的查詢可以以下列的 SQL(Standard Query Language)語言表示:

```
SELECT Part, Supplier, SUM(Sales) AS Total Sales
FROM R
GROUP BY Part, 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。

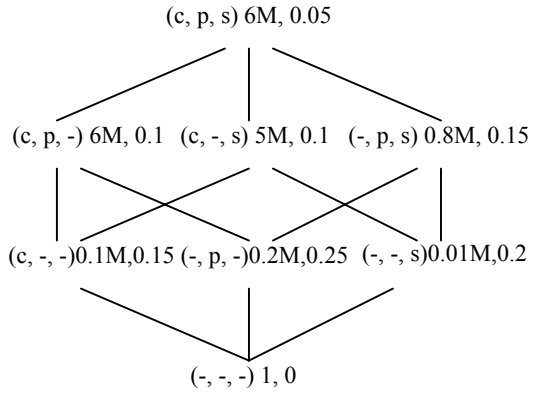


圖二、資料方體的例子。



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格構(lattice)，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 $D_1 \le D_2$。其意義為，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格構。

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，稱之為實體化。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為最短。此一最佳化的問題就稱之為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-hard 的問題。為了便於描述之後的演算法，我們將此問題正式定義如下。

假設所考慮的資料方體的所有可能的子方體為 $D_1, D_2, \ldots, D_n$，使用者所查詢的聚合問題為 $Q_1, Q_2, \ldots, Q_m$，每個問題查詢的頻率為 $f_{Q_1}, f_{Q_2}, \ldots, f_{Q_m}$，回答查詢 Q_i 所需的查詢成本為 $C(Q_i)$，而資料倉儲可以儲存的空間為 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

$$\sum_{i=1}^{n} v_i \cdot |D_i| \le S \quad , \tag{1}$$

且 $\sum_{i=1}^{m} f_{Q_i} \cdot C(Q_i)$ 為最小 (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[11]，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為(-, p, -)，所需的成本為 0.2M。然而，若是(-, p, -)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(-, p, s)，則所需的成本變為 0.8M。

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為

$$\sum_{i=1}^{n} v_i \cdot |D_i| \le S \text{，} \qquad (3)$$

$$\text{且} \sum_{i=1}^{m} f_{D_i} \cdot C_V(D_i) \text{為最小。} \qquad (4)$$

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為 V 時，使用子方體 D_i 的成本為。

三、啟發式資料方體挑選方法

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[6, 16, 17, 18]。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[11]，並提出一貪婪演算法來解決此問題；[7, 8, 9]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；近來則有 Smith 等人提出一動態的選取方法[22]。Ross 等人[20]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 [10]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為可表示成 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。在 [23]，Theodorates 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要能回答所有的查詢問題，或使用者的查詢能經由改寫(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[24]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[14]，但未考慮索引的選取問題。於國內的研究部分，則有陳耀輝等[3]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。我們亦針對此問題，提出一植基於遺傳演算法的挑選方法，能找出比一般貪婪法更佳的解[1]。

由於視域實體化的選取問題是屬於 NP-hard 問題，目前解決此一問題主要以啟發式為主。在這節中，我們描述目前已知的幾種啟發式的挑選方法，包括 Harinarayan 等人[11]所提出的正向貪婪法(FGS, Forward Greedy Selection)、我們提出的反向貪婪挑選法(BFS, Backward Greedy Selection)[2]、Shukla 等人[21]提出的體積挑選法(PBS, Pick By Size)及我們提出的近似利益挑選法(PAB, Pick by Approximate Benefit)並比較這四個演算法的效能(effectiveness，即求解的品質)與時間複雜度(time complexity)。為簡化分析並專注這些方法的特性，在此我們皆未加入任何縮減資料方體空間的方法，如[3]。另外我們假設儲存空間小於所有子方體的總合，否則所有的子方體皆可儲存，也就無挑選的必要。

(一) 正向貪婪挑選法

正向貪婪挑選法的基本作法是由零開始，一次挑選一個子方體，直到無法再增加新的子方體為止，其關鍵在於如何在每次的挑選過程中，選擇最合適的子方體。為此，我們定義一計算每個子方體的利益函數(Benefit function)，$B_V(D_i)$如下：

$$B_V(D_i) = R_V(D_i) / |D_i| \qquad (5)$$

$$R_V(D_i) = \sum_{i=1}^{m} f_{D_i} \cdot \left(C_V(D_i) - C_{V \lhd i}(D_i)\right) \qquad (6)$$

其中 $R_V(D_i)$代表的意義為，當目前的子方體實體化向量為 V 時，將子方體 D_i 加入後，回答所有的查詢所減少的時間；$R_V(D_i)$與 D_i 大小的比值，即單位空間所減少的查詢時間，而符號 $V \lhd i$ 即表示將子方體 D_i 加入後所對應的實體化向量。然後根據此定義，計算出在實體化向量 V 中的所有的子方體的價值 $B_V(D_i)$後，利益指數最高的子方體即為應挑選的對象。此正向貪婪挑選方法詳述如下：

演算法一、正向貪婪演算法。

步驟 0: 令 $V \leftarrow 00\ldots0$。

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(5)及(6)計算所有 $v_i = 1$ 的子方體 D_i 的利益值 $B_V(D_i)$，$1 \le i \le n$。

步驟 3: 從步驟 1 的結果中，挑選利益值最大的子方體，設為 D_i。

步驟 4: $V \leftarrow V \lhd j$。

步驟 5: 跳回步驟 1。

舉例來說，讓我們考慮圖四的例子：假設空間限制為 7M，且當(c, p, s)未實體化時，假設所有需使用(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為 200M。根據式子(3.1)及(3.2)，可求得(c, p, s)、(c, -, s)、(-, p, s)、(c, -, -)、(-, p, -)及(-, -, s)等六個子方體的利益指數，如表一所示。其中"受依賴的子方體"一欄中的子方體，表示其對應的使用者查詢，必須使用其所依賴的最小的實體化子方體來回答。例如，以(c, -, s)為例，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, -, -)及(-, -, s)。因此，當(c, p, s)被選取後，原本需由基底關聯資料回答的查詢就可用(c, -, s)來回答，依此可求出其利益指數為 (200−5)*0.4/5 = 16。

表一、執行正向貪婪法前兩次挑選過程中，所挑選的子方體的利益值。

	第一次挑選		第二次挑選	
	受依賴的子方體	利益值	受依賴的子方體	利益值
cps	cps, cp-, c-s, -ps, c--, -p-, --s	32	cps, cp-, c-s, -ps, c--, -p-	26
c-s	c-s, c--, --s	16	c-s, c--	10
-ps	-ps, -p-, --s	149	-ps, -p-	100
c--	c--	300	c--	**300**
-p-	-p-	250	-p-	250
--s	--s	**99995**	--s	

(二) 反向貪婪挑選法

正向貪婪挑選法的優點是，當儲存的空間遠小於子方體總合時，能較快地找出一子方體的組合；但是顯而易見地，當儲存空間與子方體總合接近時，就需要進行較多次的挑選運算。因此，在[2]我們提出一反向的作法：假設所有的子方體皆已挑選，然後再從中一次剔除一個子方體，直到子方體總合小於或等於儲存空間為止，我們將這個方法稱為反向貪婪挑選法。和正向貪婪挑選法相似，此方法的關鍵在於如何決定該剔除那一個子方體。為此，我們定義一計算每個子方體的損害函數(Paining function)，$P_V(D_i)$ 如下：

$$P_V(D_i) = G_V(D_i) / |D_i| \qquad (7)$$

$$G_V(D_i) = \sum_{i=1}^{m} f_{D_i} \cdot \left(C_{V \rhd i}(D_i) - C_V(D_i)\right) \qquad (8)$$

其中 $G_V(D_i)$ 代表的意義為，當目前的子方體實體化向量為 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 $V \rhd i$ 即表示將子方體 D_i 移除後所對應的實體化向量。然後根據此定義，計算出在實體化向量 V 中的所有的子方體的價值 $P_V(D_i)$後，損害值最低的子方體即為應剔除的對象。此反向貪婪挑選方法詳述如下：

演算法二、反向貪婪挑選法。

步驟 0: 令 $V \leftarrow 11\ldots1$。

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(7)及(8)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 2 的結果中，挑選痛苦指數最小的子方體，設為 D_j。

步驟 4: $V \leftarrow V \rhd j$。

步驟 5: 跳回步驟 1。

舉例來說，讓我們再次考慮圖四的例子：假設已挑選的子方體向量 V 為 10111110，空間限制為 7M。另外我們假設，當(c, p, s)未實體化時，所有需使用(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，此基底關聯資料筆數同樣設為 200M。根據式子(7)及(8)，可求得(c, p, s)、(c, -, s)、(-, p, s)、(c, -, -)、(-, p, -)及(-, -, s)等六個子方體的損害值，如表二所示。其中"受依賴的子方體"一欄的意義如前所述。例如，以(c, p, s)為例，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 (c, p, -)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，依此可求出其損害值為(200 − 6) * (0.05 + 0.1) / 6 = 4.85。由表二的結果得知，所挑選的子方體為(c, -, s)及(-,

p, -)，而剩餘的實體化子方體的空間為6.91M。

表二、執行反向貪婪法前兩次挑選過程中，所挑選的子方體的損失值。

	第一次挑選		第二次挑選	
	受依賴的子方體	損害值	受依賴的子方體	損害值
cps	cps, cp-	4.85	cps, cp-, c-s	8.08
c-s	c-s	**0.02**	c-s	
-ps	-ps	0.98	-ps	0.98
c--	c--	12.25	c--	12.25
-p-	-p-	0.75	-p-	**0.75**
--s	--s	15.80	--s	15.80

(三) 體積挑選法

此挑選法為預先計算各子方體的體積大小，然後依遞增的關係，在不超過儲存空間下，由小至大一一挑選予以實體化。令S 代表儲存空間的限制，D代表所有子方體的集合，V代表被挑選中將以實體化子方體的集合。此體積挑選方法[21]詳述如下：

演算法三、體積挑選法。

步驟 0: 將集合 D 中子方體根據大小作遞增排序。

步驟 1: 令 $V \leftarrow 00\ldots0$，$j \leftarrow 1$。

步驟 2: 當$(S - |D_j|) > 0$ 時，重覆執行步驟 3 ~ 4。

步驟 3: $S \leftarrow S - |D_j|$，$V \leftarrow V \lhd j$，$j \leftarrow j + 1$。

步驟 4: 跳回步驟 2。

以圖四的子方體方格為例，子方體的排列次序為(-, -, s)，(c, -, -)，(-, p, -)，(-, p, s)，(c, -, s)，(c, p, -)，(c, p, s)。因此，若空間限制設為7M，則所選取的子方體為前五組。

(四) 近似利益挑選法

這個方法的基本構想在簡化正向或反向挑選法決定下一個實體化方體的計算時間，概念是結合正向貪婪挑選法與體積挑選法，先估算每個子方體的利益值，如下所示

$$\left(\sum_{D_j \in Anc(D_i)} |D_j| \Big/ \big(|D_i| \cdot |Anc(D_i)| \big) - 1 \right) \cdot \sum_{D_k \in Des(D_i) \cup \{D_i\}} f_{D_k} \quad (9)$$

其中 $Anc(D_i)$及 $Des(D_i)$分別表示在資料方格關係中，屬子方體 D_i 的祖先及子孫的子方體集合。

式(9)的意義表示每個子方體 D_i 被選取後所帶來的益處的預估值；因為我們無法正確知道 D_i 在何時被選取以及當 D_i 被選取時，已選取的子方體集合中何者為 D_i 的祖先及子孫，因此我們假設當 D_i 被選取時，所有在 $Des(D_i)$中的子方體的查詢時間將由 $|Anc(D_i)|^{-1} \sum_{D_j \in Anc(D_i)} |D_j|$（即子方體 D_i 的祖先子方體大小的平均值）成為 $|D_i|$，故查詢時間減少量為 $|Anc(D_i)|^{-1} \sum_{D_j \in Anc(D_i)} |D_j| - |D_i|$，再乘上所有在 $Des(D_i)$中的子方體及 D_i 本身的使用頻率總合即為所求。

根據利益估算值將所有子方體作遞減排序，最後將排序後的子方體由第一個開始累計其大小，直到累計合超過儲存空間為止。

演算法四、近似利益挑選法。

步驟 0: 根據式(9)計算個子方體的利益估計值。

步驟 1: 將集合 D 中子方體根據利益估計值作遞增排序。

步驟 2: 令 $V \leftarrow 00\ldots0$，$j \leftarrow 1$。

步驟 3: 當$(S - |D_j|) > 0$ 時，重覆執行步驟 4 ~ 5。

步驟 4: $S \leftarrow S - |D_j|$，$V \leftarrow V \lhd j$，$j \leftarrow j + 1$。

步驟 5: 跳回步驟 3。

表三、近似利益挑選法中，各子方體的利益估計值。

D_i	$Anc(D_i)$	$Des(D_i)$	利益估計值
cps	Base	cp-, c-s, -ps, c--, -p-, --s	32
c-s	Base, cps	c--, --s	9
-ps	Base, cps	-p-, --s	76
cp-	Base, cps	c--, -p-	8
c--	Base, cps, cp-, c-s	---	80
-p-	Base, cps, cp-, -ps	---	40
--s	Base, cps, -ps, c-s-	---	794

例如以圖四的子方體方格為例，子方體(c, -, s)被選取的利益預估值為((200+6)/2– 5) * (0.15 + 0.2 + 0.1) = 44；各子方體的利益預估值計算結果如表三所示。根據此結果可得到子方體的排列次序為(-, -, s)，(c, -, -)，(-, p, s)，(-, p, -)，(c, p, s)，(c, -, s)，(c, p, -)。因此，若空間限制設為 7M，則所選取的子方體為前四組。

四、效能與複雜度分析

在這節中，我們首先分析這四個演算法的時間複雜度，再分析比較各個演算法的效能。

假設執行 FGS 挑選法須執行 k 次的挑選動作(步驟 3)，每一次的挑選動作須計算所有未挑選的子方體的利益值，而每個子方體利益值的計算須對所有的查詢做計算，故所需要的時間為 $O(k{\cdot}n^2)$。BGS 的損害值的計算方式和 FGS 類似，所需時間相同，但因 BGS 是以刪除的方式反向進行，是以若同一問題 FGS 需進行 k 次挑選，則 BGS 大約需進行 $n-k$ 次刪除，所以時間複雜度為 $O((n-k)\,n^2)$。

PBS 主要的運算為排序，故所需的時間為 $O(n \log n)$。至於 PAB，我們假設資料方體格構為超立方體(Hypercube)，且資料方體具有 d 個維度，故 $n = 2^d$。由此假設可以推出，對某一具有 i 個維度的子方體 v，其祖先及子孫的數目為

$$|Anc(v)| = 2^{d-i} - 1，\ |Des(v)| = 2^i - 1。$$

故計算所有子方體的預估利益需時

$$\begin{aligned}\sum_{i=1}^{d}\begin{bmatrix} d \\ i \end{bmatrix}\left(2^{d-i}-1+2^i-1\right) &= 2\sum_{i=1}^{d}\begin{bmatrix} d \\ i \end{bmatrix}2^i - 2\sum_{i=1}^{d}\begin{bmatrix} d \\ i \end{bmatrix} \\ &= 2\cdot 3^d - 2^{d+1} \\ &\cong O(3^{\log n}).\end{aligned}$$

此處 $\begin{bmatrix} d \\ i \end{bmatrix}$ 表示組合函數，即由 d 個相異物件中選取 i 個不同物件的組合總數。

由上述的分析結果可以了解就速度而言，PBS 最佳、其次為 PAB，最後為 FGS 及 BGS，而 FGS 及 BGS 孰優孰劣端視 k 的大小而定當 k 小於 $n/2$ 時，FGS 有較佳的速度，反之則為 BGS。

接下來我們分析這四個演算法的效能。在[11]中，Harinarayan 等人證明 FGS 所求得的解所減少的查詢成本(即利益值)不會少於最佳解所減少的查詢成本的 63%。在[21]中，Shukla 等人也證明，若子方體格構為**限定體積超立方體**[1](Size-restricted hypercube)，則 PBS 所求得的解與 FGS 相同，換言之，FGS 所求得的解所減少的查詢成本亦不會少於最佳解所減少的查詢成本的 63%。此處我們證明 BGS 及 PAB 亦有相同的效能。

定理 1、對任何的子方體格構，BGS 選定刪除的子方體所增加的查詢成本不會多於最佳解所增加的查詢成本的 1.58 倍。

證明：令 $v_1, v_2, \ldots, v_k$ 表 BGS 所選定需刪除的 k 個子方體，而 $w_1, w_2, \ldots, w_k$ 為最佳方法所選定需刪除的子方體，a_i 表 $v_1, v_2, \ldots, v_{i-1}$ 子方體已刪除的情況下，刪除子方體 v_i 所增加的查詢成本，b_i 為 $w_1, w_2, \ldots, w_{i-1}$ 子方體已刪除的情況下，刪除子方體 w_i 所增加的查詢成本，$1 \le i \le k$。此外，令刪除 $v_1, v_2, \ldots, v_k$ 所增加的查詢成本為 A，$A = \sum_i a_i$，而刪除 $w_1, w_2, \ldots, w_k$ 所增加的查詢成本為 B，$B = \sum_i b_i$。

仿照[11]中的證明方式，我們可以導出

$$\frac{B}{A} \ge 1 - \left(\frac{k-1}{k}\right)^k$$

當 $k \to \infty$，$(\frac{k-1}{k})^k$ 趨近於 $1/e$（e 表示自然數），所以 $A/B \le e/(e-1) = 1.58$。 ■

引理 2、對任何的子方體格構，BGS 求得實體化的資方體所減少的查詢成本不會少於最佳解所減少的查詢成本的 63%。

至於 PAB 的效能，假設資料方體格構為限定體積超立方體，且$|u| \ge (k+3)\,|v|$，$k = |\text{children}(u)|$，我們可以得到下列結果。

定理 3、對任一滿足限定體積超立方體性質的資料方體格構，且$|u| \ge (k+3)\,|v|$ ($k = |\text{children}(u)|$)，PAB 所求得的解所減少的查詢成本不會少於最佳解所減少的查詢成本的 63%。

證明：若我們能證明 PAB 所求出的解與 PBS 相同，則可證明 PAB 與 PBS 有同等的效能。欲證明此點，只要證明體積愈小的子方體其近似利益值愈大即可，因為 PAB 所得到的子方體排列次序與 PBS 相同。

令一具有 i 個屬性的子方體 v 位於資料格構中的第 i 層($0 \le i \le d$)，而 l_i 表示 v 與其最大的兒子方體大小的比值，則 v 的兒子方體數目為 i，$l_i \ge i + 3$。此外，若位於資料格構中第 0 層的方體大小為 s，則位於第 i 層的資

[1] 所謂的限定體積超立方體為構成超立方體中的任二節點 u 及 v (u 為 v 的父節點且 u 不為根節點)，節點 u 的大小至少為節點 v 的大小的 $k+1$ 倍($k = |\text{children}(u)|$)，即 $|u| \ge (k+1)\,|v|$。例如圖四的格構即為限定體積超立方體。

料方體的大小皆至少為 $s\sum_{j=1}^{i} l_j$ 。

根據超立方體的定義， v 的子孫數目為 $2^i - 1$，而位於第 j 層的祖先數目為 $\binom{d}{j}$ ，$i+1 \le j \le d$。因此，v 的近似利益值 $AB(v)$為

$$AB(v) \ge \left(\frac{\sum_{j=i+1}^{d} \binom{d}{j} s \prod_{r=1}^{j} l_r}{2^{d-i} s \prod_{r=1}^{i} l_r} - 1 \right) \times 2^i$$

$$= 2^{2i-d} \sum_{j=i+1}^{d} \binom{d}{j} \prod_{r=i+1}^{j} l_r - 2^i$$

現考慮另一子方體 w。若 w 與 v 位於同一層，且 w 的大小小於 v，由上式可得知 w 的近似利益值大於 v。

若 w 位於第 i–1 層的子方體，則 w 的近似利益值 $AB(u)$為

$$AB(u) \ge 2^{2i-d-2} \sum_{j=i}^{d} \binom{d}{j} \prod_{r=i}^{j} l_r - 2^{i-1}$$

$$= \frac{l_i}{4} 2^{2i-d} \sum_{j=i+1}^{d} \binom{d}{j} \prod_{r=i+1}^{j} l_r + \frac{l_i}{4} \binom{d}{i} 2^{2i-d} - 2^{i-1}$$

因為 $l_i \ge i+3 \ge 4$，所以 $AB(u) \ge AB(v)$。

綜合以上所述，PAB 求得的子方體排列次序為依資料方體大小作遞增排序。 ■

上述的結果似乎證明這四種啟發式的挑選方法是**近似演算法**(approximate algorithms)[12]－保證求得解的品質與最佳解的差距在一定的範圍內，然而事實卻非如此。這個結果的衡量標準(criterion)－利益值，實際上並非真正的查詢成本。因此，即使利益值高於最佳利益值的 63%，實際的查詢成本卻可能遠高於最佳查詢成本。例如，令 T 代表未進行任何子方體實體化的總查詢成本，假設最佳的挑選方法求得的實體化子方體可將查詢成本降為 $0.01T$，而某一貪婪挑選法求得的實體化子方體只能將查詢成本降為 $0.2T$，為最佳解的 20 倍，然而若比較利益值可發現

$(T - 0.2T) / (T - 0.01T) = 81\%$。

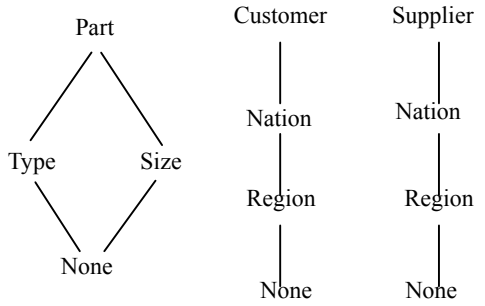
事實上針對資料方體的選取問題，Karloff 及 Mihail [15]已證明除非可在多項式時間內解出之問題的集合等於不可在多項式時間內解出之問題的集合(即 P = NP)，否則不存在有可在多項式時間內完成的近似演算法。他們並進一步證明當格構中的子方體數目 n 大於 20 且為 4 的倍數時，FGS 所求得的查詢成本至少為最佳解的 n/12 倍。

五、實驗比較

為了解這四個演算法實際的效能與執行效率，我們以 TPC-D 決策支援測試庫[19]進行一連串的實驗。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，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。

表四、TPCD 中，由 customer，part，supplier 三個屬性所組成的資料方體中所有的子方體。

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



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

我們從這資料庫中挑選三個維度來組成測試用的資料方格(data cube)，分別為顧客(Customer, c)、零件(Part, p)、及供應商(Supplier, s)等，而為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將有 64 種，其相關的大小列述於表四，其中所有的屬性都以第一個英文字母來表示，"-"的符號為 none 的意思，空間單位為 K(1000)資料筆數。

至於各個子方體的使用頻率的組合，我們考慮了三種模式：第一種是所有的子方體的使用頻率皆相同，且都大於零；第二種是由隨機產生介於 0~1 間的亂數所組成的組合；第三種則是設成子方體的使用頻率和其大小成線性反比。另外我們將儲存空間的限制，分別設為子方體總合的 10%、20%、30%、40%、50%、60%、70%、80%及90%。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，我們則將其視為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為資料方體中最大的子方體的 3 倍，即 18M。

我們首先比較這四種挑選法的效能，結果如圖六、圖七及圖八。由圖中我們可以很清楚的發現，不管頻率為何，正向與反向貪婪法在查詢成本上都優於其他兩種，而正向與反向貪婪法則互有領先。當空間限制較小時(< 40%)，正向貪婪法在子方體的使用頻率為線性時表現較佳，而反向貪婪法則再其他兩種頻率時表現較佳；當空間限制較大時，兩種方法的表現差不多。這是因為空間限制較大時，兩種方法所得出的解差異不大。

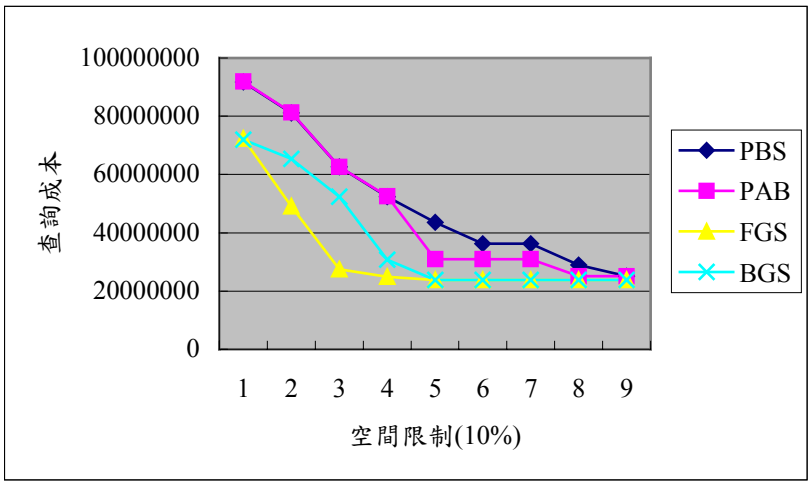
另外，就體積挑選法與近似利益挑選法兩者相比，可以發現近似利益挑選法的表現都優於體積挑選法。

在第四節中，我們曾提及此處所分析的四種演算法雖然就利益值而言都是近似演算法，但實際上就問題真正的查詢成本而言，這四個演算法未必能得到近似最佳解。在[1]中，我們提出一以遺傳演算法為基礎的挑選方法，在與正向貪婪法的比較實驗中，我們就發現兩者的查詢成本差距最大達到 3~4 倍之多。

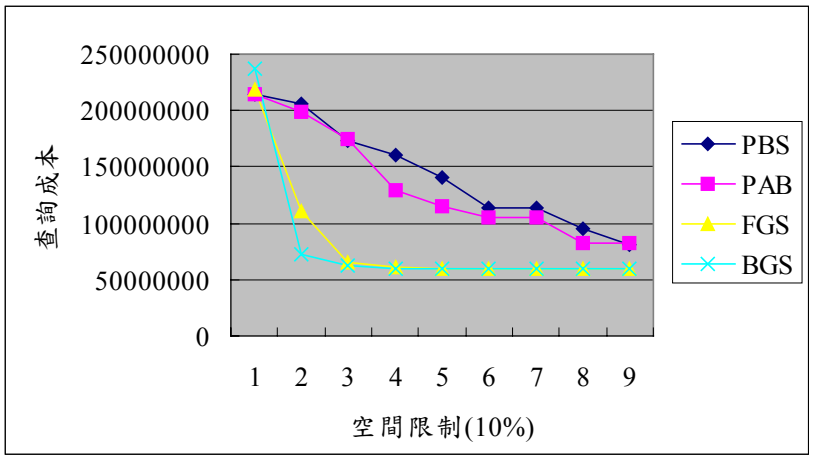
接下來我們比較這四個演算法的執行時間，此處僅顯示子方體使用頻率皆相同時的時間，結果如圖九所示，其他兩種頻率的結果與此相似。基本上實驗的結果與第四節的理論分析相符合，正向與反向貪婪挑選法的速度最慢，其次為近似利益挑選法，最快為體積挑選法，正向與反向挑選法兩者的優劣與空間限制有關，而體積挑選法與近似利益挑選法的速度極為相近。

五、結論

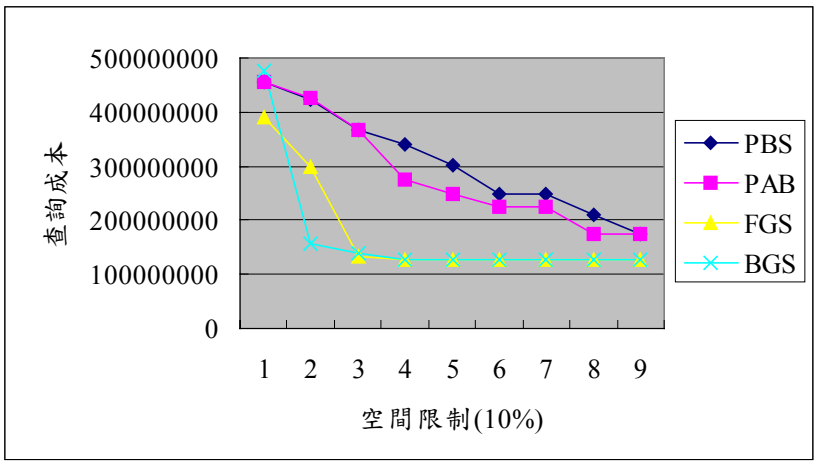
在本論文中，我們針對資料倉儲中資料方體的挑選問題，在儲存空間的限制條件下，分析比較幾種啟發式挑選法的效能與速度，並進行實驗驗證。綜合這些分析比較的結果，我們認為在實際設計資料倉儲系統時，如何選取欲實體化的資料方體應可考慮



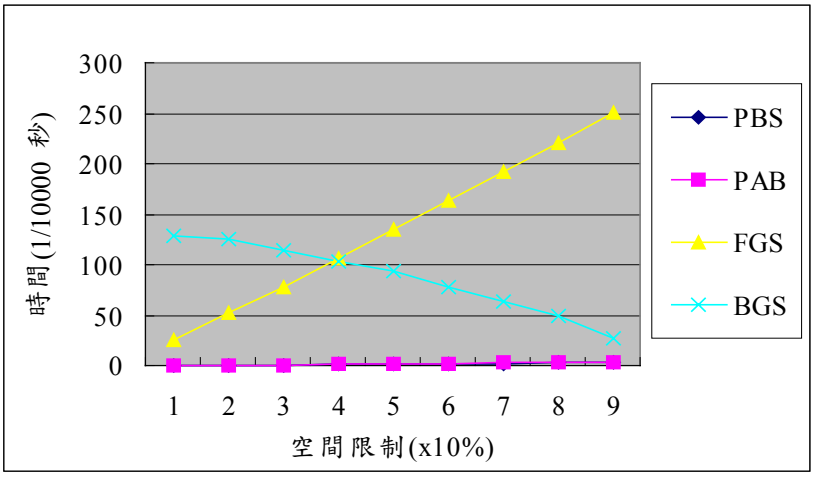
圖六、當各子方體的使用頻率成線性時，各挑選法在查詢成本上的比較。



圖七、當各子方體的使用頻率為亂數時，各挑選法在查詢成本上的比較。



圖八、當各子方體的使用頻率皆相同時，各挑選法在查詢成本上的比較。



圖九、當各子方體的使用頻率皆相同時，各挑選法執行時間的比較。

下列的綜合性做法：初次建立資料方體時，　　可根據預估的全體子方體的大小與硬碟的空

間的比率決定使用正向或反向貪婪挑選法，此階段著重的是所挑選的實體化子方體的效能，盡量降低查詢成本；往後在來源資料增加或更新下，需更新實體化的資料方體時，則可以使用較快速的體積挑選法或近似利益值挑選法，以期能一方面在短時間內完成資料方體的更新工作，另一方面又能兼顧查詢成本的降低。

由於啟發式的挑選法在質的方面離最佳解能有相當的差距，況且以正向或反向挑選方法的時間複雜度 $O(k \cdot n^2)$，在子方體數目較大時，所需的執行時間還是相當長，未必符合實際的需求，未來我們將專注於尋找更快速且效能更好的挑選方法。

六、參考文獻

[1] 林文揚與郭義中，“遺傳演算法於資料倉儲中構建資料方體之應用”，八十八年全國計算機會議論文集，第一輯，第241-248 頁，1999。

[2] 林文揚與郭義中，“應用於線上分析之資料方體的雙向貪婪挑選法”，第五屆資訊管理研討會，高雄，臺灣，2000。

[3] 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。

[4] E. Baralis, S. Paraboschi, E. Teniente, “Materialized view selection in a multidimensional database,” in Proceedings of the 23rd VLDB Conference, Athens, Greece, 1997.

[5] S. Chaudhuri and U. Dayal, “An overview of data warehouse and OLAP technology,” in ACM SIGMOD Record, Vol. 26, pp. 65-74, 1997.

[6] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, and K. Shim, “Optimizing queries with materialized views,” in Proceedings of ICDE, pp. 190-200, 1995.

[7] 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.

[8] H. Gupta, “Selection of views to materialize in a data warehouse,” in Proceedings of ICDT, pp. 98-112, 1997.

[9] H. Gupta,V. Harinarayan, A. Rajaraman, and J.D. Ullman, “Index Selection for OLAP,” in Proceedings of ICDE, pp. 208-219, 1997.

[10] H. Gupta and I.S. Mumick, “Selection of views to materialize under a maintenance cost constraint,” in Proceedings of International Conference on Database Theory, Jerusalam, Israel, 1999.

[11] V. Harinarayan, A. Rajaraman, and J.D. Ullman, “Implementing data cubes efficiently,” in Proceedings of ACM SIGMOD ‘96, pp. 205-216, 1996.

[12] E. Horowitz and S. Sahni, Fundamentals of Computer Algorithms, Computer Science Press, Inc., 1978.

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

[14] M. Jarke, “Common subexpression isolation in multiple query optimization,” Query Processing in Database Systems, pp. 191-205, 1984.

[15] H. Karloff and M. Mihail, “On the complexity of the view-selection problem,” in Proceedings of ACM PODS, Philadelphia, USA, pp. 167-173, 1999.

[16] A. Levy, A.O. Mendelson, Y. Sagiv, and D. Srivastava, “Answering queries using views,” in Proceedings of ACM PODS, pp. 95-104, 1995.

[17] P.-A. Larson and H.Yang, “Computing queries from derived relations,” in Proceedings of the 1st VLDB Conference, pp. 259-269, 1985.

[18] D. Quass, A. Gupta, I.S. Mumick, and J. Widom, “Making views self maintainable for data warehousing,” in Proceedings of PDIS, 1996.

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

[20] K.A. Ross, D. Srivastava, and S. Sudarshan, “Materialized view maintenance and integrity constraint checking: trading space for time,” in Proceedings of ACM SIGMOD, pp. 447-458, 1996.

[21] A. Shukla , P. M. Deshande and J. F. Naughtion, “Materialized View Selection for Multidimensional Datasets,” in Proceedings of the 24th VLDB Conference, New York, USA, 1998.

[22] J. R. Smith, et al, “Dynamic assembly of views in data cubes,” in Proceedings of

ACM PODS, pp. 274-283, 1998.

[23] D. Theodoratos and T. Sellis, “Data warehouse configuration,” in Proceedings of the 23rd VLDB Conference, Athens, Greece, 1997.

[24] J. Yang, K. Karlapalem, and Q. Li, “Algorithm for materialized view design in data warehousing environment,” in Proceedings of the 23rd VLDB, Athens, Greece, 1997.