Xa-tree：在 OLAP 上有效支援範圍查詢的多維度索引架構

李建億、陳科學、李育強
國立台南師範學院資訊教育研究所
E-mail：leeci@ipx.ntntc.edu.tw

一、摘　要

資料倉儲是結合資料庫與決策支援的產物，作為資料庫的後端作業。儲存於資料倉儲中的各種資料，有助於提供線上分析處理(On Line Analysis Processing，OLAP)，允許透過資料庫查詢語言(SQL)，提供即時的整合性資訊。範圍查詢(Range Query)是 OLAP 中的重要分析工具之一，R*a-tree 以 R*-tree 為基礎的索引結構，其內部節點存放子樹的特徵值以增進範圍查詢效能。然而，以 R-tree 為基礎的樹狀索引結構，其內部節點在高維度下有嚴重 overlap 的問題，使得無效搜尋次數增加，影響查詢效率。X-tree 有著接近於 R*-tree 的建構方法，且改善 R*-tree 於中高維度時嚴重的 overlap 問題。因此，我們提出 Xa-tree 以結合 X-tree 及 R*a-tree 在內部節點存放子樹特徵值的特性，以獲取更佳的範圍查詢效率。實驗結果證明：改良自 X-tree 的 Xa-tree，在中維度空間資料的範圍查詢，擁有高於 R*a-tree 的效率，比 R*a-tree 更適於中維度空間的資料範圍的總集查詢。

關鍵詞：線上分析處理、關聯式資料庫、範圍查詢

二、前言

資料倉儲是結合資料庫與決策支援的產物，作為資料庫的後端作業。儲存於資料倉儲中各種資料，有助於提供線上分析處理(On Line Analysis Processing，OLAP)。允許透過資料庫查詢語言(SQL)，提供即時的整合性資訊，以給管理人員快速且有效地進行決策。

OLAP 依照實作方法，又可區分為關聯式線上分析處理(Relational On Line Analyze Processing，ROLAP)和多維資料線上分析處理(Multi-dimension On Line Analyze Processing，MOLAP)兩種。ROLAP 是運用關聯式表格，以 star 或 snowflake schema 的方式建立，可於傳統資料庫型態實作。而 MOLAP 則是陣列型態。

線上分析處理的目的是提供快速且整合性的資料，幫助使用者以最有效率的方式，觀察趨勢、判斷機會、進行分析和預測未來。資料方體(Data Cube)[5]則是描述的 OLAP 中多維屬性聚集化(Aggregation)架構的模型。對查詢範圍內的加總(sum)或平均(average)之查詢需求，統稱之為總集查詢(aggregate)查詢。本文將著重於 ROLAP 上總集範圍查詢架構的研究。

目前應用於高維度資料的索引結構，在 R-tree 系列方法中，有 R-tree[4]、R+-tree[13]、R*-tree[1]、TV-tree[8]、X-tree[2]、SS-tree[14]、SR-tree[7]以及改良資料結構，用以提升範圍查詢效率的 R*a-tree[6]等。而非 R-tree 系列的方法計有 Grid File[10]、Bitmap index[9]、K-D-B tree[11]及其相關的方法等。Grid File 的優點是建構方式相對簡單，但是儲存空間使用效率不佳；Bitmap 則適用於屬性少維度低的場合。而 K-D-B tree 及其及其相關的方法則擅長於點查詢的表現。

在第三章中，將探討與本研究相關的文獻。將針對 R-tree 及相關索引法與 X-tree 索引法作重點介紹。在第四章中，則將提出針對 X-tree 做為改良型樹狀資料結構，以提升總集查詢效率的方法。第五章是實驗結果報告。第六章則是結論。

三、文獻探討

多維空間索引結構目前廣為應用的除 R-tree 系列外，尚有 K-D-B tree、Grid file、Bit-map index 及其相關方法。R-tree 與 K-D-B tree、Grid file 及其相關方法相比，其優點是不需要經由額外的方式，轉換原始空間資料以儲存，因而提供了較佳的資料叢聚性[2]。和 Bit-map 相比，R-tree 更適用於多維度、區域集合密度大、update 頻繁的場合[12]。R-tree 的運用廣泛，其觀念直接或間接衍生了許多變形樹，如 R+-tree、R*-tree、X-tree、SS-tree、SR-tree 及 R*a-tree 等。

Guttman 於 1984 年提出 R-tree，其以 B-tree 為基礎，由單維點搜尋(Point Access Method)，擴充至多維空間搜尋(Spatial Access Method)。在圖 2.1 中，一群實線矩型的物件(由 d 至 n)的集合是 R-tree 的葉節點，A、B、C 則是以最小涵蓋範圍所決定出的目錄層。其所成的階層式架構如圖 2.2 所示。R-tree 的結構與 B-tree 大同小異，其節點(node)分為葉節點(leaf node)和非葉節點(non-leaf node)。R-tree 和 B-tree 的最大差異是：B-tree 的葉節點直接指向資料點本身，R-tree 則是指向資料所處的「範圍」；這項重要差異使 R-tree 可以處理二維以上形態的資料。R-tree 的幾個重要特徵於下述：

1. 葉節點的表示法為(I，*tuple-identifier*)，非葉節點表示法是(I，*child-identifier*)。I 代表有限維度的集合。前者指向實體資料所存在的範圍，後者則是指向下一層子節點的集合範圍。
2. 設其最大擁有非空節點的分支度為 M，最小

為 m，且 $m \leq M/2$。

3. 對於非葉節點而言，存在有一最小含蓋區域(*Minima Bounding Range，MBR*)，以最小範圍涵蓋所有的所屬子節點。
4. 樹根(root)至少有兩個分支。而對於每個節點而言，其擁有的分支度數在 m 和 M 之間。(樹根除外)
5. 所有的葉節點都在同一個階度上。

R-tree 具體的概念圖，則如圖 2-2 與圖 2-3 所示：

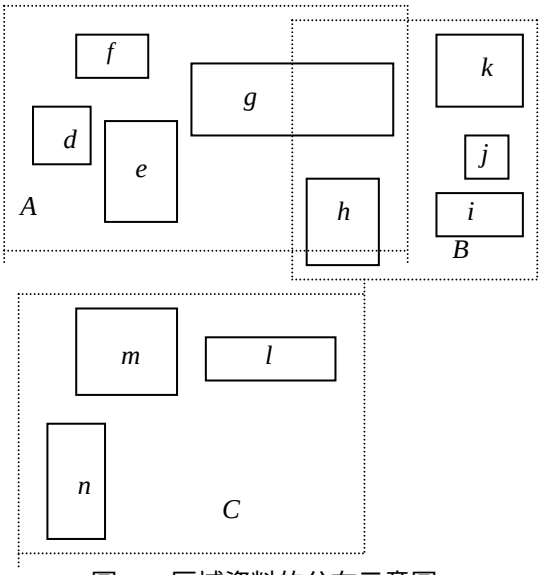

圖 2.1 區域資料的分布示意圖

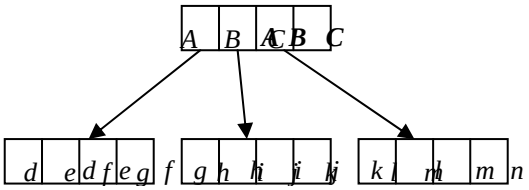

圖 2.2 R-tree 的階層式架構

由於空間涵蓋的特性，R-tree 容忍節點範圍重疊(overlap)的現象。Overlap 是指：兩個或以上的鄰近節點所涵蓋的區域，存在重疊的情況。但是 Overlap 會影響 R-tree 搜尋效能；為了減低 overlap，R-tree 藉由最小涵蓋區域(MBR)的決定來降低 overlap 的產生。

R+-tree 是將 R+tree 內部節點 overlap 部分切割，以避免 overlap。

R*-tree 針對 R-tree 做更進一步的改良。其最主要的改良有幾點：

1. 在 R*-tree 裡，樹的形成和變動除了考慮最小涵蓋範圍外，亦考量最小包圍周長。
2. 其實驗得知 R-tree 最佳的樹狀分支度比。當最小分支度(m)為 0.4 倍的最大分支度(M)時，R-tree 的效能達到最佳狀態。
3. 在新增葉節點時， R*-tree 進一步考量最小重疊成本的因素(Minimum overlap cost)。
4. 採用拓撲分裂法(topological split)。不同於 R-tree 常用的二次分裂。
5. 強制性重新插入(force reinsert)的機制。

Berchtold 指出：R-tree 在高維度(維度在 5 以上)應用環境下，其拓撲分裂法將近於失效，而中間目錄層的重疊幾乎超過 90%；過多的重疊覆蓋會造成物件無效搜尋的路徑增加。在低維資料的應用場合，無效路徑尚不明顯；但在高維資料，則嚴重影響查詢的效率。而通常高度的重疊覆蓋，多半發生在資料密度高的區域。在最嚴重的情況下，高度的重疊覆蓋，將使得部份範圍查詢效能表現，接近於整顆樹的搜尋。因此 X-tree 提供了 supernode 的觀念：對於高度目錄重疊的區間，不如將產生重疊的目錄合併，而成為一個大型中間節點稱之為 supernode。

其 supernode 的產生機制為：

1. 設定 overlap 門檻值。當節點一分為二時，檢查 overlap 是否超過此門檻值。若低於門檻，則採取和 R*-tree 相同的拓普分裂法。
2. 若依照拓普分裂法後結果，將高於 overlap 門檻值，將進行 overlap-free split。
3. 若 overlap-free split 分裂的結果，一側節點所含括之 entry 數低於最小分支度的最低標準，則傳回分裂失敗的訊息。並將原父節點進行倍數擴張，使其成為 supernode。

當一個 supernode 因為目錄合併而產生時，其父節點亦重新決定，擴展為 supernode 以包涵之。圖 2.3 是 R*-tree 目錄重疊覆蓋狀況，圖中 g、h 兩目錄具有高度的 overlap；圖 2.4 則是 X-tree 的 supernode 示意圖：在 g、h 合併成為 supernode *S2* 後，為了保持高度平衡的特性，在計算最小涵蓋面積(MBR)以後，選擇了 A 將其 supernode 化成為 *S1*，以包涵 *S2*，降低目錄層重疊情況。supernode 只有在拓撲分裂法和 Minimal overlapping spilt 皆無法得到低於門檻值的解才會產生[2]。

R*a-tree 為提升範圍查詢的效率，改良 R*-tree 目錄節點，在原有的指標節點外，增加一欄位存放該節點的特徵值。如增加 count 或 summary 欄位，以記錄其葉節點的個數，或所有值的總和，在進行範圍查詢時，只需拜訪目錄層，即可知道所要的答案；而不需深入拜訪每個葉節點，使樹的搜尋深度減少，加速整合性資料的查詢[6]。為了其它方面應用的需求，特徵值可以是其它模式，如子樹所含資料之標準差(standard deviation)或變異數(variation)，以利進行統計分析。

SS-tree、SR-tree 等則是處理空間特徵向量的樹狀結構，其延伸 R*-tree 最大面積和最小周長的觀念，修改最小涵蓋範圍為矩形的作法，改以圓形涵蓋所屬的節點。但是依照 Data cube 的環境，屬性特性呈彼此垂直的關係，是以 SS-tree 和 SR-tree 的作法並不適合。

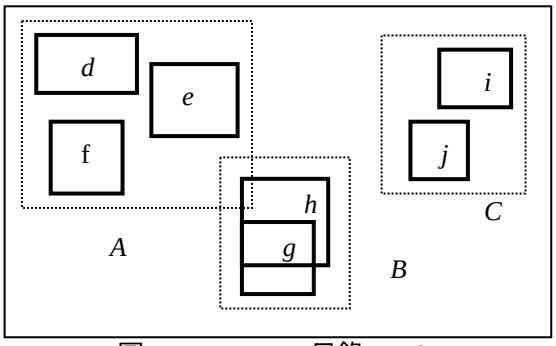

圖 2.3、 R*-tree 目錄 overlap

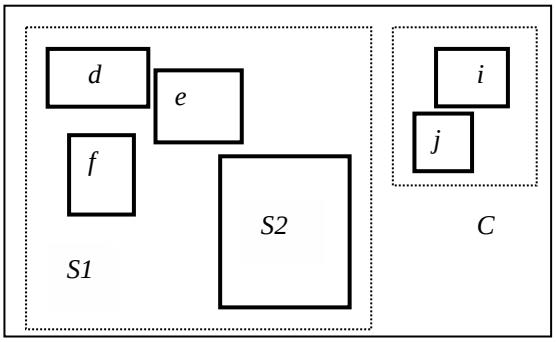

圖 2.4、 X-tree 改進目錄的 overlap

四、ROLAP 索引結構的改進

ROLAP 最主要的著眼點，是提供使用者得到快速整合性的資料，如總和(sum)或次數(count)等資訊，以幫助進行決策。在此條件下，總集查詢(Aggregate Query)成為主要查詢的行為。

X-tree 為了減低高維空間下，R*-tree 中間目錄節點的高度重疊影響效率，因而有 supernode 的出現。而為了能提供更好的範圍查詢效率，R*a-tree 在中間層的目錄節點上，增添了擺放特徵涵數值的欄位。因此，我們提出結合 X-tree 與 R*a-tree 方法，以提升範圍查詢的效能，稱之為 Xa-tree。

圖 4.1 及 4.2 分別為高維環境下，R*-tree 和 X-tree 與查詢窗(query window)的關係圖。其中淺虛線代表 root 下第一層目錄層，深黑虛線代表欲查詢之範圍。由圖可以看出，由於 R*-tree 的目錄層 overlap 嚴重，故無可避免的需進行較繁複之判斷，並往來於各目錄層之間。X-tree 的 overlap 有效減少，各目錄層子樹範圍明確，所以在進行總集查詢時，能夠較 R*-tree 更確定搜尋目標節點，減少無效目錄層的誤判與往返路徑。在 R*-tree 的資料結構增加了子樹特徵值而成為 R*a-tree，並修改範圍查詢的指令後，固然可以加快效率。但仍無法避免因先天上高度 overlap 所帶來的困擾。R*-tree 在五維空間以上的資料，因分裂法運作失效而導至大量的 overlap，代表在相同的資料量下，其擁有較多的內部節點；在進行相同型態的資料查詢時，必需較 X-tree 多出更多的路徑和拜訪節點。而 X-tree 在這點上取得優勢：在雙方的資料結構皆做同樣的改進時，Xa-tree 仍可因為較低的 overlap，子樹搜尋較 R*a-tree 更明確，進而擁有較高的效率。此外，在提升總集查詢效率的同時，Xa-tree 也延續了 X-tree 在點查詢上的優於 R*-tree 的能力，從而兼顧總集查詢與點查詢的效能需求。

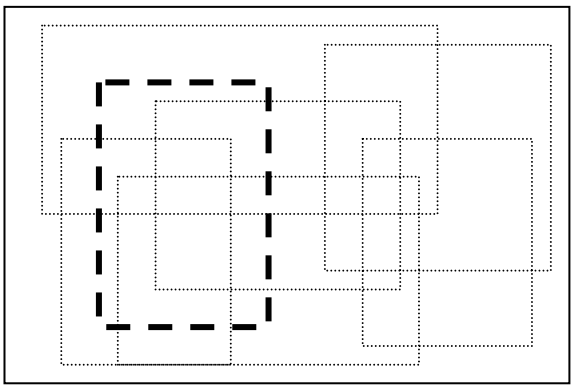

圖 4.1 Query window 之於 R*-tree 目錄層 overlap

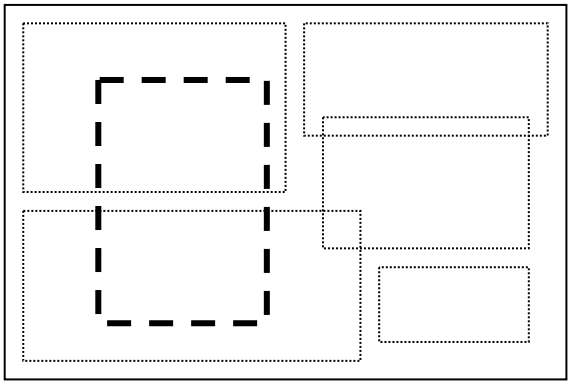

圖 4.2 Query window 之於 X-tree 目錄層

隨著目錄層節點增加子樹特徵值，範圍查詢法也需配合修改，才能夠提升效率。Xa-tree 的範圍查詢演算法，同 R*a-tree，如圖 4.3 所示。

```
function RangeSum(page p, rectangle query)
sum:=0;
if p is leaf-node
   for all data_entry ∈ p
      if (data_entry ∩ query) ≠ 0
         sum = sum + data_entry;
else
   for all dir_entry ∈ p
      if dir_entry ⊆ query
         sum = sum + dir_entry.sum;
      else
         if (dir_entry ∩ query) ≠ 0
            sum = sum + RangeSum(dir_entry.succ,query);
return (sum)
```

圖 4.3、總集範圍查詢演算法

五、實驗結果

為驗證本方法之可行性，我們以 X-tree 為基礎，實作 Xa-tree，語言為 GNU C++，作業系統為 Linux3.4。執行硬體平台為 AMD K6II-500，128Mb RAM。分別以 2 維、4 維、6 維、8 維及 10 維等五種維度，用電腦模擬 uniform 分配資料各十萬筆，建構和測試 Xa-tree，R*a-tree，X-tree，和 R*tree 等四種不同樹狀結構，研究其建樹成本，和範圍查詢的表現。

分別以建樹成本和範圍查詢效率，作為效能分析的項目。在建樹成本上，我們記錄了建樹時間及內部節點個數，表 5.1、5.2 及 5.3 四種樹狀結構在建樹時間及內部節點數量，在較低維度時幾乎是相同的。原因是在低維度空間，R*-tree 的拓撲分裂法仍能有效運作，故 X-tree 並未進行更複雜的 overlap-free 分裂法，以降低目錄節點之間的 overlap，也沒有任何目錄節點進化成 supernode。而 Xa-tree 之於 X-tree，和 R*a-tree 之與 R*-tree 兩者的改變相同，在資料結構的差異，皆僅於中間目錄層增加其下子樹的特徵值。在這裡，僅於每個中間節點差一個 4 bytes 的子數特徵值。而在 10 維時，Xa-tree 及 X-tree 的內部節點明顯減少，但卻升高 CPU 計算時間。

表 5.1 各種樹狀結構建樹的 CPU 計算時間(秒)

維度	Xa-tree	X-tree	R*a-tree	R*-tree
2	35.02	34.87	35.16	34.83
4	63.3	61	62.59	61.76
6	93	91.94	92.03	90.82
8	118.64	116.59	117	115.63
10	203.71	201.49	146.77	144.66

表 5.2 各種樹狀結構建樹的內部節點數

維度	Xa-tree	X-tree	R*a-tree	R*-tree
2	22	22	22	22
4	56	56	56	56
6	120	120	123	123
8	197	197	196	196
10	120	120	289	289

表 5.3 各種樹狀結構建樹的葉節點數

維度	Xa-tree	X-tree	R*a-tree	R*-tree
2	1055	1055	1055	1055
4	1447	1447	1447	1447
6	1797	1797	1797	1797
8	2172	2172	2142	2142
10	2566	2566	2567	2567

總集範圍查詢效能表現上，在各種不同維度下，進行範圍大至整個資料庫，小至資料庫空間的十分之一，共十種不同大小的範圍查詢各一百次，記錄不同樹狀結構範圍查詢的表現。實驗結果記錄 CPU 計算秒數和節點存取次數。因 CPU 運作的速度快，為得到較為具體的數字，故表列之 CPU 計算時間，為一百次查詢結果之累計。而節點存取次數，則為不同總集查詢範圍，進行一百次隨機查詢後的整數平均值。

由圖 5.1 及圖 5.2 得知，R*a-tree 及 Xa-tree 在範圍查詢時的 CPU 計算時間明顯低 R*-tree 及 X-tree。R*a-tree 及 Xa-tree 的 CPU 計算時間十分相近。

在一般的情況下，Xa-tree 的效率表現，與 X-tree 接近；R*a-tree 的效率表現，與 R*-tree 接近。由圖 5.3 及圖 5.4 得知，R*a-tree 及 Xa-tree 在範圍查詢時的存取節點數明顯低 R*-tree 及 X-tree，且 Xa-tree 更低於 R*a-tree。運用範圍總合查詢所需要存取的節點數愈少，也就是查詢時所需要拜訪的節點數愈少就可以得到所要的回應，意謂著效能愈高。

在低維環境(2 到 4 維)下 Xa-tree 和 Ra-tree 的範圍查詢效能方面，並無差異性存在。基於建樹方法的分析，我們認為這是可以預期的結果。此二種結構在低維度時具有同樣的資料量，和相同數目的中間目錄層，效率自是無所差異。

圖 5.5 顯示 8 維時在各種查詢範圍下，Xa-tree 優於 R*a-tree。10 維時則更明顯，如圖 5.6。

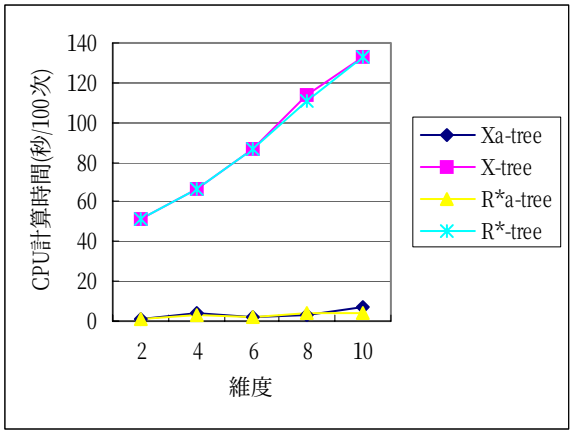

圖 5.1 查詢範圍 90%各維度 CPU 計算時間

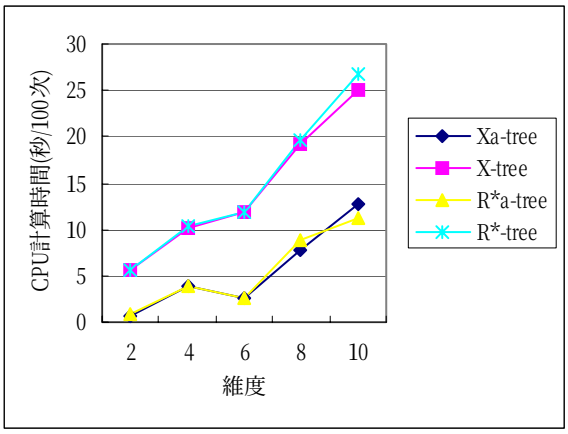

圖 5.2 查詢範圍 10%各維度 CPU 計算時間

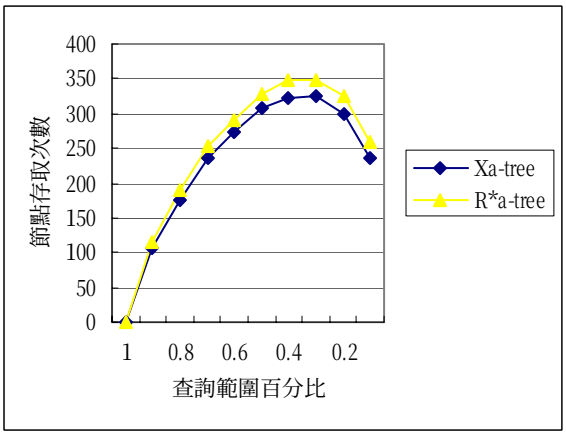

圖 5.5 8 維時各種查詢範圍的節點存取次數

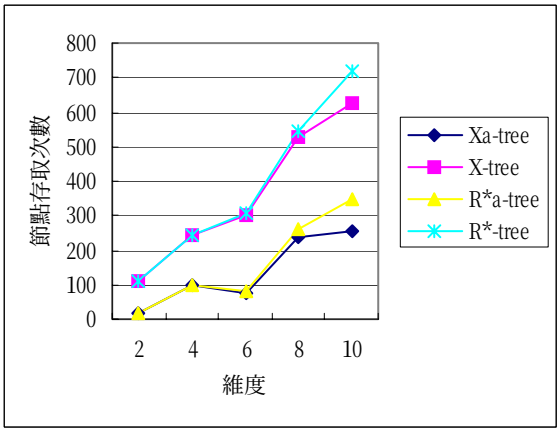

圖 5.3 查詢範圍 90%各維度節點存取次數

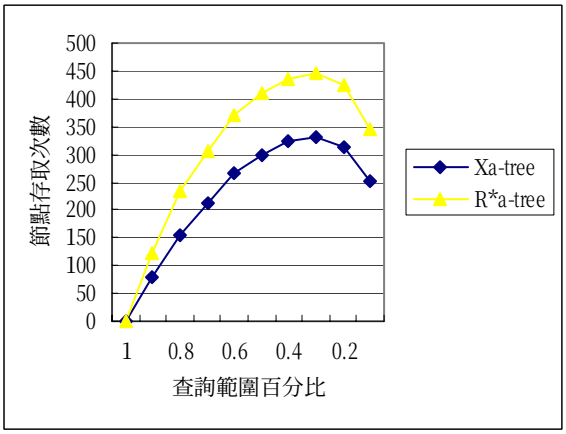

圖 5.6 10 維時各種查詢範圍的節點存取次數

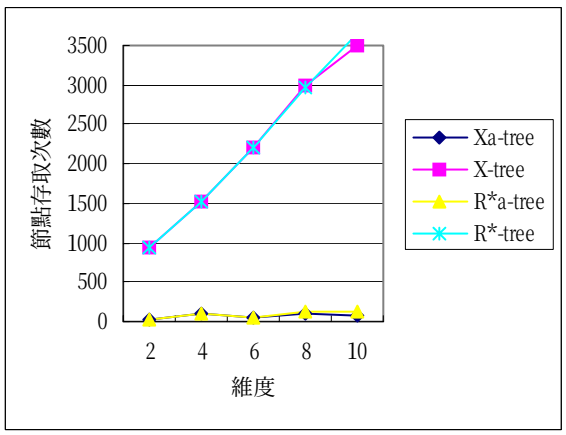

圖 5.4 查詢範圍 10%各維度節點存取次數

六、結論

綜合本研究的實驗結果，我們發現：改良的 Xa-tree 資料結構，其目錄節點增添子樹特徵變數(如 summary 或是 count)，在均勻分佈情況下，於中維度資料(六維到十維)，Xa-tree 在總集範圍查詢上普遍優於 X-tree、R*a-tree 與 R*-tree。由於四種樹在建樹時的成本，並未出現明顯差異；故為得到較好的效率，使用 Xa-tree 是值得考率的選擇。

因此，在中維度資料以下環境的 ROLAP 索引，建議可以 Xa-tree 取代 R*a-tree 實作，可獲致一般性效率的提升。

針對總集範圍查詢所改良的 Xa-tree，在中低維度均勻分部環境進行一般範圍的總集查詢時，效率較其它三種樹(X-tree，R*a-tree，R*-tree)的效能有所提升。未來將進一步對其他類型的資料做驗證，以及更高維度的實驗，並擴充應用至其他總集函數。

七、參考文獻：

[1] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, Bernhard Seeger (1990), The R*-tree: An efficient and robust Asses Method for point and Rectangles. *Proceedings ACM-SIGMOD International Conference on Management of Data, Atlantic City, NJ,* 322-331.

[2] Stefan Berchtold, Daniel A. Keim, Hans-Peter Kriegel (1996), The X-tree: An index structure for High-dimensional data, *Proceedings of 22th International Conference on Vary Large Data Base, Mumbai(Bombay), India*. 28-39.

[3] Himanshu Gupta , Venky Harinarayan, Anand Rajaraman1 (1997), Index selection for OLAP, *International Conference on Data Engineering, Birmingham, U.K.*, 208-219.

[4] Antonin Guttman (1984), R-trees：A dynamic index structure for spatial searching, *Proceedings ACM-SIGMOD International Conference on Management of Data, Boston, MA*, 47-57.

[5] Venky Harinarayan , Anand Rajaraman, Jeffery D. Ullman (1996), *Implementing Data Cubes Efficiently ， Proceedings ACM-SIGMOD International Conference on Management of Data*, 205-216.

[6] Marcus Jurgens, Hans-J. Lenz (1998), The R*-a tree: An improved R*-tree with materialized data for supporting range queries on OLAP data, *DEXA Workshop*. 186-191.

[7] Norio Katayama, Shin'ichi Satoh (1997), The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries. *Proceedings ACM-SIGMOD International Conference on Management of Data*. 369-380

[8] King-Ip Lin, H. V. Jagadish, Christos Faloutsos (1994), The TV-Tree: An Index Structure for High-Dimensional Data. *VLDB Journal* 3(4),517-542

[9] P.O' Neil and G. Graefe (1995), Multi-table joins through bitmapped join indices. *SIGMOD Record, 24(3), Sept*, 8-11.

[10] J. Nievergelt, H. Hinterberger, and K. C. Sevcik (1984), The Grid File: An Adaptable, symmetric multikey file structure, *ACM Transaction on Database systems. 9(1)*, 38-71.

[11] J. T. Robinson. (1981), The K-D-B tree: A search structure for large multidimensional dynamic indexes, *Proceedings ACM-SIGMOD Conference, Ann Arbor, Michigan, June,* 10-18.

[12] S. Sarawagi (1997), Indexing OLAP Data. *IEEE Bulletin on Data Engineering*, 20(1), March , 36-43.

[13] Timos Sellis, Nick Roussopoulos, Christos Falousos (1987), The R+-tree: A dynamic index for multi-dimensional objects, *Proceedings of 13th, International Conference on Vary Large Data Base, Brighton, England.* 507-518.

[14] David A. White, Ramesh Jain (1996), Similarity Indexing with the SS-tree. *International Conference on Data Engineering.* 516-523