submit for poster session

應用於子維度天際線查詢的全新 G-tree

(A Novel G-tree for Subspace Skyline Query)

黃璽合
國立成功大學資訊工程所
kid863@hotmail.com

李　強
國立成功大學資訊工程所
leec@mail.ncku.edu.tw

摘要

天際線搜尋演算法在近期的資料庫研究領域中越趨重要。給定一組在多維度資料庫中的資料集，天際線搜尋會回傳那些不被其他點支配的資料點。在實務上，需要採用天際線搜尋的資料庫，通常都會提供多組候選維度，然而使用者只會對少部分有興趣。因此，通常都會根據多種維度的子集合進行查詢，而這類型的查詢稱之為子維度天際線查詢。使用傳統的天際線演算法來直接處理這種查詢是非常沒有效率的。有許多額外的演算法與架構被用來改善搜尋效率；然而，這些修改都會增加計算成本或是必須增加資料儲存空間。本篇論文提出一個基於 Gaussian Function 的全新索引模型，用來增強子維度天際線查詢的效率。實驗模擬結果展現了新的索引樹在子維度中找出天際線點的高效性。

關鍵詞：資料庫、天際線、樹狀結構。

Abstract

The skyline search algorithm has recently emerged as an important technique in database research. Given a set of data points in a multi-dimensional database, such queries return points that are not “dominated” by any other point. In practice, databases that require a skyline query usually provide numerous candidate dimensions, of which users are interested in only a few. As a result, queries are issued regarding various subsets of the dimensions and such queries are called subspace skyline queries. Using the conventional skyline algorithm to process these queries directly can be extremely ineffective. Additional algorithms and architectures have been added to improve search efficiency; however, such modifications can increase computational costs or necessitate an increase in data storage capacity. This paper proposes a novel index model based on a Gaussian function to enhance the performance of subspace skyline queries. Simulation results demonstrate the efficacy of the proposed tree in locating skyline points within a subspace.

Keywords: *Database, Skyline, Tree Structure.*

一、簡介

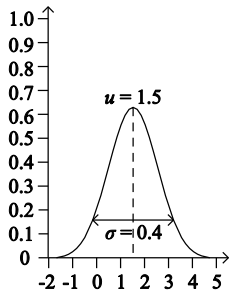
子集合天際線查詢(subspace skyline queries) [2][5]是天際線查詢的重要子議題。假設整個資料集有 d 個維度，這個查詢只會依據其中 n 個維度($n < d$)的數值進行天際線查詢。舉例來說，一個實際的旅館資料庫常會儲存旅館五個以上的資訊。例如，房間價格、房間大小、餐廳價格、旅館星級、餐廳星級、旅館到海灘的距離及旅館到機場的距離。然而，一個使用者可能僅對房間價格及到機場的距離感興趣。另一個使用者卻可能只對房間價格、飯店等級及到機場的距離感興趣。由於這兩位使用者並無考慮資料庫中的全部資訊，所以依資料庫中全部維度為基礎所找出來的天際線資料點(傳統的天際線查詢)並不能滿足這兩位使用者的需求。因此，我們必須開發新的演算法來處理子集合天際線查詢。

本論文會設計一個新的索引樹架構,稱為高斯樹(Gaussian tree)來改善子集合天際線查詢的效率。高斯樹中同一節點的資料點，不管在所有維度中，還是子集合維度中，都是非常接近的。因此，高斯樹可以克服 R-tree 在處理子集合天際線查詢時所遇到的問題。亦即，當我們以一個依據資料庫全部維度數值所建立的高斯樹，來找子集合維度中的天際線資料點時，演算法將不需要讀取過多的資料點，所以查詢的效率會改善。此外，由於高斯樹與 R-tree 同為樹之架構，所以高斯樹可以直接應用於傳統的天際線演算法中，而不需要修改原本的演算法或加入額外的演算法。最後，我們提供了數個模擬來驗證使用高斯樹來處理子集合天際線查詢的有效性。

二、設計方法、技術與步驟

(一)　高斯樹架構

submit for poster session



圖一、高斯函數的例子

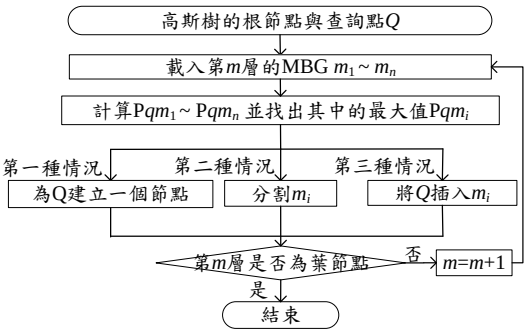
高斯樹是用如圖一的高斯函數(Gaussian Function)所建立的，這個函數可被寫為[1]

$$f(x)=\frac{1}{\sigma\sqrt{2\pi}}e^{-(x-u)^2/2\sigma^2} \qquad (1)$$

其中 u 代表高斯函數的平均值(圖一中鐘形函數的中心)，σ 則代表高斯函數的標準差(圖一中鐘形函數的寬)。

(二) 高斯樹建立演算法

在建立高斯樹時，演算法每次會將一個資料點加入高斯樹中。每個被加入的資料點則會經過如圖二所示的處理程序。



圖二、高斯樹建立演算法的流程圖

假設要被插入的資料點為 $Q\,(q_x,q_y)$，則演算法會從根節點開始，一層一層檢查 Q 該被插入哪個節點中。這個過程將會持續到 Q 被插入任一個葉節點為止。假設要檢查的層為 m，且此層的節點為 m_1 到 m_n，則演算法會計算 Q 落在 m_1 到 m_n 中的機率，Pqm_1 到 Pqm_n。假設 Pqm_i 為這些機率中最大的，則依據 Pqm_i 與使用者給定的 MBG 邊界數值 α，我們會進行以下的動作。

第一種情況，$Pqm_i < \alpha$：在這個情況下，Q 與第 m 層的節點皆不相關，所以我們會在第 m 層建立一個以 Q 為中心的 MBG m_e。在 m_e 被建立後，演算法會在第 m-1 層中找尋適合插入 m_e 的父節點。若有找到，演算法會建立 m_e 與該父節點的關係，否則，演算法會在第 m-1 層建立一個以 m_e 為中心的父節點。

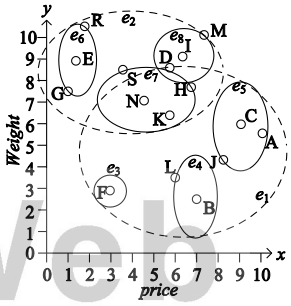
第二種情況，$Pqm_i \geq \alpha$，且 m_i 達到容量上限：在這個情況下，Q 會先被插入 m_i 中，m_i 則會再分裂成 m_{i1} 與 m_{i2} 兩個節點，其中，m_{i1} 的中心點為 m_i 的中心點，m_{i2} 的中心點則為 m_i 中距離中心點最遠的資料點。接著，m_i 內及其周圍的資料點則會被重新分配到各自最相關的節點中。

第三種情況，$Pqm_i \geq \alpha$，且 m_i 尚未達到容量上限：在這個情況下，Q 會直接被插入 m_i 中，並結束這個動作。

表一為使用圖三進行子集合天際線查詢的詳細步驟說明。

表一、使用高斯樹進行子集合天際線查詢的例子

步驟	堆疊	動作	S
1	Φ	取出根節點	Φ
2	e_1, e_2	拆開 e_1	Φ
3	e_3, e_2, e_4, e_5	拆開 e_3	Φ
4	F, e_2, e_4, e_5	移動 F 到 S	{F}
5	e_2, e_4, e_5	拆開 e_2	{F}
6	e_6, e_4, e_7, e_5, e_8	拆開 e_6	{F}
7	G, e_4, E, R, e_7, e_5, e_8	移動 G 到 S	{F, G}
8	e_4, E, R, e_7, e_5, e_8	拆開 e_4	{F, G}
9	B, L, E, R, e_7, e_5, e_8	移動 B 到 S	{F, G, B}
10	L, E, R, e_7, e_5, e_8	從堆疊中刪除 L	{F, G, B}
11	E, R, e_7, e_5, e_8	從堆疊中刪除 E	{F, G, B}
12	R, e_7, e_5, e_8	從堆疊中刪除 R	{F, G, B}
13	e_7, e_5, e_8	從堆疊中刪除 e_7	{F, G, B}
14	e_5, e_8	從堆疊中刪除 e_5	{F, G, B}
15	e_8	從堆疊中刪除 e_8	{F, G, B}
16	Φ	結束	{F, G, B}



圖三、高斯樹的例子

在第一步中，演算法會取出高斯樹根節點的子 MBG e1 及 e2，並將這兩個 MBG 依據他們座標值的總和放入堆疊中。在第二步中，演算法會首先取出堆疊的第一個元素 e1。然而因

submit for poster session

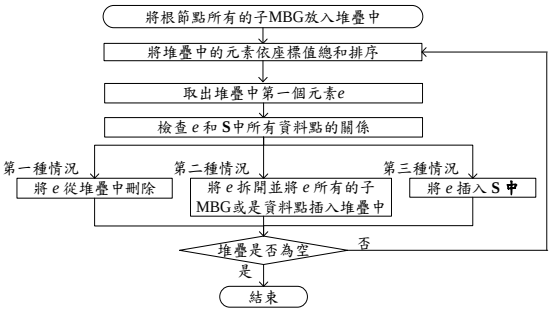
為 S 為空，所以 e1 不可能被任何資料點支配。這個情況符合第二種結果，所以 e1 會被拆開，且 e1 內的子 MBG e3、e4 及 e5 會依據他們座標值的總和放入堆疊中。第三步到第 16 步會重複類似的步驟。然而因為第四步跟第 10 步為第三種結果與第一種結果的例子，所以以下我們會進一步解釋這兩步。

在第四步中，演算法會取出堆疊的第一個元素 F 進行處理。然而，因為此時 S 為空，所以, F 不可能被任何資料點支配。這個情況符合第三種結果，所以我們會將 F 放入 S 當中，並在堆疊中刪除 F。

在第 10 步中，演算法會取出堆疊的第一個元素 L 進行處理。然而，從圖三中我們可知 L 會被 S 中的 F 所支配。這個情況符合第一種情況，所以 L 不可能成為子集合天際線查詢的答案，而直接從堆疊中刪除。

(三) 將高斯樹應用於子集合天際線查詢中

將高斯樹應用於子集合天際線查詢中的演算法流程圖如圖四所示。



圖四、高斯樹處理子集合天際線查詢的流程圖

這個演算法使用了三種資料結構，包含了一棵高斯樹、一個堆疊(heap)及一個表格(list)。堆疊是用來儲存演算法的暫時資訊，這些資訊可能為高斯樹內的 MBG 或是資料點。以下為了方便解釋，我們將堆疊內所儲存的資訊稱為元素(element)。表格則是用來儲存子集合天際線查詢的答案。以下我們將所有子集合天際線資料點的集合則稱為 **S**。這個演算法可以分為兩個部分，如圖四所示。第一個部分會將高斯樹根節點下的所有內部節點放到堆疊中。在堆疊中，所有的元素會以他們座標值的總和做升冪排列，其中，若元素為一節點，則其座標值總和將會等於此節點左下角座標值的總和[3]。在第二個部分，演算法每次會檢查堆疊中最上面的元素 *e* 是否會被 **S** 中任一子集合天際線資料點支配。這個檢查會有以下三種可能的結果。

第一種結果：如果 *e* 被 **S** 中任一個子集合天際線資料點支配，則 *e* 會從推疊中直接刪除。

第二種情況：如果 *e* 並沒有被 **S** 中任一個子集合天際線資料點支配，且 *e* 為一個 MBG，則 *e* 會被拆解且其內的 MBG 或是資料點將被放入堆疊中。

第三種情況：如果 *e* 並沒有被 **S** 中任一個子集合天際線資料點支配，且 *e* 為一個資料點，則 *e* 為一個子集合天際線資料點。所以 *e* 將被放入 **S** 中並從堆疊中刪除。

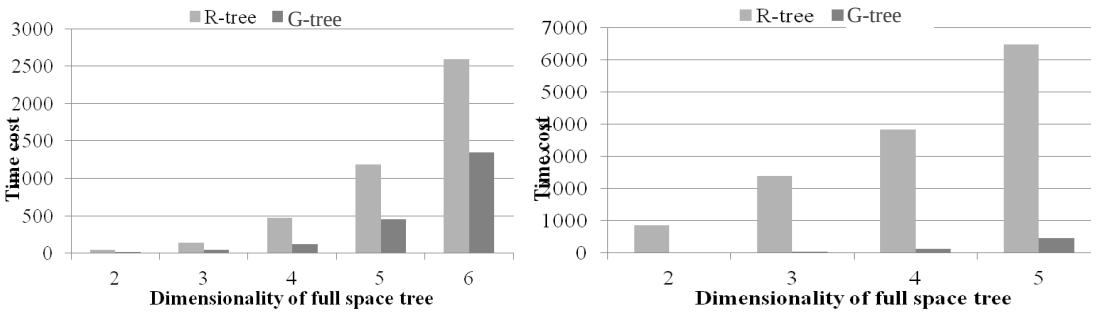
當堆疊中不再包含任何元素時，這個程式將會終止。此時，所有 S 內的資料點即為子集合天際線查詢的答案。

三、實驗模擬

本章使用了數個實驗來證明使用高斯樹進行子集合天際線查詢的有效性。模擬的資料集是天際線問題中最常用的資料集，分別是 independent 資料集及 anti-correlated 資料集 [3] [4]。一般來說，Independent 資料集會被視為天際線問題中普通的情況。anti-correlated 資料集則會被視為最差的情況，因為他具有最多的資料點。本模擬會首先建立具有六個維度的 independent 資料集 及 anti-correlated 資料集，且每個資料集具有一百萬筆的資料。接著，我們會分別對這兩個資料集建立 R-tree 與高斯樹，並使用這兩棵樹來找出 2 到 5 維的子集合天際線查詢答案。本章的程式是以 MATLAB® 開發，模擬電腦則使用 2.93GHz 的 Intel Core2 Duo CPU 並配合上 2GB 的記憶體，作業系統則為 Microsoft Windows XP。

圖五為使用 R-tree 與高斯樹找出二到五維子集合天際線查詢結果所需要的時間。圖五(a)為 independent 資料集的結果，圖五(b)為 anti-correlated 資料集的結果。在這兩張圖中，我們可以看到不管是使用 R-tree 還是高斯樹，演算法所需的時間都會隨維度上升。這是因為子集合天際線資料點的數量會隨維度增加而成長[3]。

submit for poster session



(a) Independent 資料集　　(b) Anti-correlated 資料集

圖五、R-tree 與高斯樹在不同子集合維度數目下進行天際線查詢所需花費的時間

在這兩張圖中，我們也可看到，R-tree 時間曲線隨維度上升的程度會比高斯樹快。因為演算法使用在六維建立的 R-tree 來找二到五維的子集合天際線資料點時，R-tree 同一節點中的資料點在子集合維度中可能變的不相近。因此，即使演算法只讀取與天際連結線交集的節點出來處理。演算法還是會讀取許多遠離天際連結線的資料點(不可能成為子集合天際線資料點的資料點)出來處理，進而大幅增加演算法的 I/O 花費。然而高斯樹卻沒有此缺點。對高斯樹來說，不管在哪一種子集合維度中，同一節點內的資料點總是相近的。所以當演算法只讀取與天際連結線交集的節點出來處理。只有與天際連結線相近的資料點(有可能成為子集合天際線資料點的資料點)會被讀入，所以演算法的 I/O 花費及時間花費可被降低。這個模擬也驗證了使用高斯樹來進行子集合天際線查詢的效率。方程式請置於欄位之中央位置並以小括號編號(1)、(2)等，並與本文內容上下保持一行之間距。

四、結論

這篇論文提出了一個新的架構來提升子集合天際線查詢的效率。本篇論文一開始闡述使用 R-tree 來進行子集合天際線查詢的缺點。接著，我們介紹了高斯樹的架構及建立演算法。同時，我們也說明了如何使用高斯樹來進行子集合天際線查詢。最後，實驗模擬則證實了使用高斯樹來進行子集合天際線查詢的效率。在我們未來的工作中，我們會嘗試將高斯樹應用於其他種類的天際線查詢，並證明在大部分的天際線問題中，使用高斯樹將比使用 R-tree 有效。

五、致謝

This work was supported by NSC under the Grant NSC 100-2221-E-006-250-MY3.

參考文獻

[1] Y. C. Chen and C. Lee, “G-tree: A Novel Index Structure for Subspace Skyline Query,” in *Proc. International Conference on Advancements in Information Technology* , 2013.

[2] W. Jin, A. K. H. Tung, M. Ester, and J. Han, “On Efficient Processing of Subspace Skyline Queries on High Dimensional Data,” in *Proc. International Conference on Scientific and Statistical Database Management*, 2007.

[3] D. Papadias, Y. Tao, G. Fu, and B. Seeger, “An optimal and progressive algorithm for skyline queries,” in *proc. ACM Special Interest Group on Management of Data Conference*, 2003.

[4] K. L. Tan, P. K. Eng, and B. C. Ooi, “Efficient progressive skyline computation,” in *Proc. International Conference on Very Large Databases*, 2001, pp. 301-310.

[5] Y. Tao, X. Xiao, and J. Pei, “SUBSKY: Efficient computation of skylines in subspaces,” in *proc. International Conference on Data Engineering*, 2006, pp. 65-74.