R-tree 一個多物件刪除的指令

陳靖國*
朝陽科技大學資訊管理系
jkchen@cyut.edu.tw

黃仁偉
朝陽科技大學資訊管理系
s9114628@cyut.edu.tw

摘要

R-tree 是一個有效率的空間物件的索引結構。以往有若干論文探討 R-tree 單一物件的存取操作，但少有同時對多個物件進行存取。如果有若干個物件要刪除，傳統 R-tree 是將物件一個接著一個從資料庫中刪除，因而產生許多不平衡(underflow)的節點，導致這些不平衡節點內的剩餘物件必須重新插入(reinsert)R-tree。由於一再的重新建構 R-tree 會導致資料庫的效率下降，所以我們提出一個存取指令(operation)稱為空間刪除(spatial-delete)，可以將一個資料庫中多個物件一次同時刪除，亦即將一群 R-tree 物件從資料庫中整批刪除，並調整資料庫 R-tree 之架構。當許多物件被刪除後，我們利用合併節點的手法來重新建構 R-tree 以解決不平衡的問題，進而達到整體最佳化(global optimization)的結果。用一個多物件刪除的指令來取代多個單一物件刪除的指令，其目的就是為了減少不平衡葉節點內剩餘物件重新插入的次數，避免 R-tree 結構不斷重整的情況一再發生而影響資料庫系統之產能(throughput)。最後，我們提出一個實際完整的例子來說明所提的空間刪除指令之功用與提高效率之證明。

關鍵字：重物件存取、空間刪除、R-tree

1. 簡介

近年來由於GIS（Geographical Information System）和CAD（Computer Aided Design）等之發展，使得空間物件的處理變得愈來愈重要。有許多空間物件的動態索引(dynamic index)可以加速資料搜尋。一般而言，多維度的空間資料可分成 zero-size 物件和 non-zero-size 物兩類。Robinson的K-D-B tree[9]和Kumar的G-tree [8]是屬於zero-size 物件的索引結構，而Nievergelt *et al.* 的Grid files[7]和Sevcik與koudas的Filter tree[12]則為non-zero-size 物件的結構。在Gaed與Gunther論文[3]中對於各種空間物件的存取方法有詳細的介紹，其中以R-tree及其家族在空間資料的存取上使用度最廣[1, 2, 4, 5, 10, 11]。在以往使用R-tree結構來對空間物件處理的文獻中，我們發現大多以提升資料存取速度[2, 4]、減少dead space[11, 12]或改善空間的利用[1, 5]為主，少有論文提及如何同時對R-tree多個物件進行存取的操作。然而，我們可能為了某些特定的需求而必須一次刪除多個物件，傳統的R-tree刪除動作一次只能刪除一個物件。假如每次只刪除一個物件，那我們將花費相當多的時間來完成這個工作，因為有很多時間必須用在對該資料庫所對應的R-tree進行一再地重整而導致系統產能明顯受到影響。以下舉例說明。

圖 1(a)和圖 1(b)分別表示若干個實線矩形所代表的空間物件與有陰影的矩形所代表的搜尋視窗 SW(search window)的分佈及對應的 R-tree 架構圖。假使要將與 SW 重疊的物件 r6 和 r8 刪除，

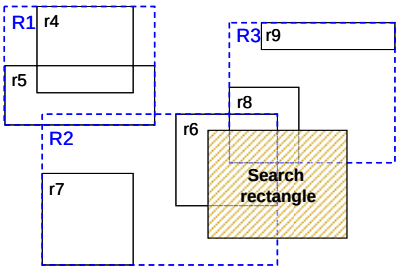


圖 1(a). 原始空間物件群與search window的分佈。

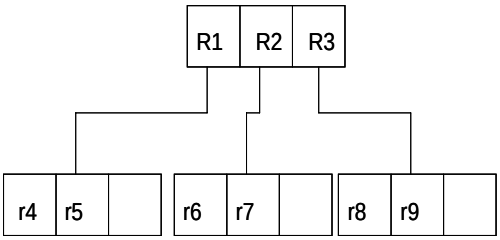


圖 1(b). 原始R-tree架構圖。

根據原先 R-tree 刪除的作法是先找出物件 r6 將其刪除，此時便造成節點 R2 的不平衡，而 R2 所包含的物件 r7 則必需取出再重新插入於 R1 節點中並刪除 R2，如圖 1(c)和圖 1(d)。

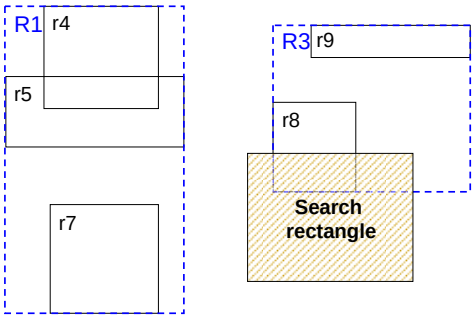


圖 1(c). 刪除物件 r6 後的物件群。

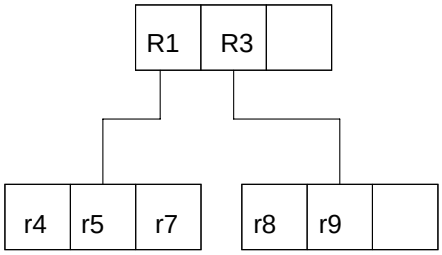


圖 1(d). 刪除物件 r6 後的 R-tree 架構。

接著找出物件 r8 後將其刪除，造成節點 R3 的不平衡，需將物件 r9 從 R3 中取出再重新插入 R1 並刪除 R3。此時，根節點只剩 R1 一個子節點，所以 R1 便成為新的根節點，如圖 1(e)和圖 1(f)。物件 r9 重新插入根節點時，因根節點已無

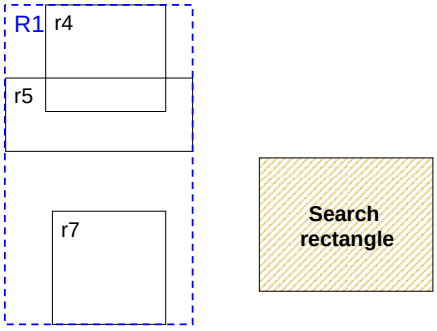


圖 1(e). 刪除物件 r8 後的物件群。

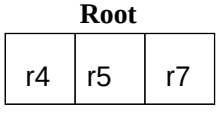


圖 1(f). 刪除物件 r8 後的 R-tree 架構。

多餘空間容納 r9，所以根節點需進行分裂成兩個節點來容納物件 r4、r5、r7 和 r9，並產生一個新的根節點來包含舊根節點 R1 及其分裂出來的新節點 R1`，如圖 1(g)和圖 1(h)。整個刪除的過程

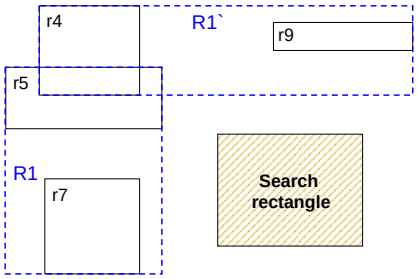


圖 1(g). 最終的物件群。

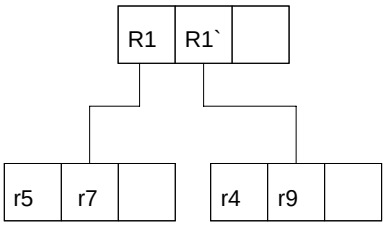


圖 1(h). 最終的 R-tree 架構。

中，多次的物件刪除與重新插入使得 R-tree 高度縮小又變大，加上不斷地調整 MBR(Minimum Bound Rectangle)大小而消耗不少時間。為了解決此問題以提升系統產能，我們提出一個叫做空間刪除的指令，可以有效地從若干個葉節點中一次刪除多個物件，而所剩餘的物件不會有重新插入的動作，讓每個節點只會有一次的合併動作，而且只有一次的 MBR 調整，減少 R-tree 調整結構的次數而提高系統產能。

2.相關工作

R-tree[4]是一種類似 B-tree 的多維度樹狀結構索引，由中間節點和葉節點所構成，所有索引的資訊都儲存於葉節點其中包含了指向實際物件資料的指標。B-tree 是傳統一維數值或文字型態的索引資料結構，而 R-tree 則是多維的空間物件的索引資料結構。R-tree 節點之間沒有次序關係，這與傳統 B-tree 節點之間有一定的次序情況，有顯明的不同。R-tree 中的節點是由若干個 entry 所組成，而 entry 則代表某一物件或子節點的相關資訊。每個 entry 包含一個 MBR 和一個指標。包含實際物件資訊的 entry 都儲存於葉節點中，其格式為(*I*，*obj-id*)，*I* 為 MBR 代表含蓋該空間物件的最小矩形，*obj-id* 代表某個空間物件所在的位址。中間節點的 entry 則可以表示成(*I*，*child-pointer*)，*I* 為 MBR 代表含蓋某個子節點的最小矩形，*child-pointer* 則是指向這個子節點的位址。基本上，父節點所包含的範圍會完全包含子節點所包含的範圍。在 R-tree 中，假設 *M* 代表

的是每個節點所能擁有的最大量 entry 數，也叫做最大分支數(maximum branch)，而 *m* 代表最小分支數，一個 R-tree 必須滿足以下的幾個特性。根節點至少要有二個子節點，除非它是個葉節點。每個非根節點的 entry 數必須介於 *m* 和 *M* 之間。所有的葉節點都必需在同一階層。

3.空間刪除指令

在這個章節中，我們將描述空間刪除指令如何從 R-tree 同時刪除多個空間物件。空間刪除指令分成二個階段。第一個階段，從 R-tree 中找出與 SW 有重疊的物件。第二個階段，將找出的物件一次整批刪除，接著使用合併節點的手法來處理所剩餘的物件調整 R-tree 架構。

3.1 空間刪除指令的第一階段

我們假設存在一個 SW，功能如同[4]可以含蓋若干個物件。首先必需找出所有與 SW 有重疊的物件對應之記錄，搜尋方式也與[4]之搜尋方式類似。我們利用一個名為 SQ (Search Queue)的佇列來儲存這些與 SW 有重疊的節點或物件所對應之 entry。從 R-tree 的根節點開始，判斷所含的 entry 所指之子節點或物件是否與 SW 有重疊。若有則將該 entry 存入 SQ 中。接著再從 SQ 中依序取出第一筆 entry，判斷該 entry 所指之節點或物件是否有重疊到 SW。若有，則將該 entry 存入 SQ 中。重複上述的 entry 取出、檢查等動作直到 SQ 中所有指向節點的 entry 記錄都處理完為止。最後，SQ 中將存有若干個 entry 指向與 SW 重疊的物件。

3.2 空間刪除指令的第二階段

在這個步驟中我們要將第一步驟中所找到的物件從R-tree中刪除。原先的R-tree在進行物件刪除時是使用重新插入的方式[4]，當物件刪除後的葉節點其所容納的entry數無法滿足R-tree節點所限制的最小數目時(亦即不平衡)，則此節點內的所有的entry需被取出、重新插入R-tree。若是有大量物件被刪除時，物件重新插入的次數便會暴增，造成R-tree不斷地進行重新建構，在執行效率上便會明顯地降低，所以傳統R-tree的刪除動作在一次刪除多個物件的應用上效率是不佳的。我們使用合併節點的方法重新建構R-tree來解決葉節點不平衡的問題，不但可以提高處理速度而且還可以達到整體最佳化的結果。合併節點的方法分成四個步驟來完成。(1)根據SQ中的entry記錄刪除該entry所對應的物件。(2)使用一個叫做ROUTE的表格來記錄R-tree所有剩餘物件的完整路徑，以便在進行R-tree重整或向上調整MBR時可以找出到相關葉節點的祖先節點。ROUTE中的每一列資料(叫做Spread)由兩個欄位CurrentN和ChildN組成。CurrentN表示R-tree中的某個節點，而ChildN則是表示CurrentN所包含的某個子節點或物件。走訪R-tree所有剩餘物件將組成各個路徑的Spread資料記錄於ROUTE中。(3)我們假設R-tree在刪除後所剩餘的物件數為R_{obj}，葉節點所能容納的最大entry數為N_{cap}。因此我們可以計算出容納所有剩餘物件所需的葉節點個數為$N=\lceil R_{obj}/N_{cap}\rceil$。將物件分成$N$個群組，並分別將這些剩餘物件分配到這$N$個葉節點中。分配法則是從物件中選出相距最遠的$N$個entry為各葉節點的第一筆記錄。接著以MBR最小擴張範圍為原則來分配剩餘的物件，使每個葉節點容量達半滿以上，直到分配完所有物件。(4)當最底層合併完成後便逐一向上調整各個祖先節點的MBR，直到某個節點之MBR不需調整或抵達根節點為止。

4.實例說明

以下使用一個實例來說明 R-tree 為何需要多個物件刪除的功能。有一都市開發計劃欲規劃一個工業區促進土地利用率。而工業區所含蓋的範圍內可能會重疊到數個建築設施。因此，我們需一次將被重疊的建築設施物件從資料庫中刪除。假設有一記錄都市各建築設施分佈的 R-tree 索引，葉節點所儲存的物件代表各建築設施的位置，其架構如圖 2(a)和圖 2 (b)所示。圖 3 則是說明物件和 SW 重疊的關係，而 SW 則為工業區所規劃的範圍。首先，從 R-tree 的根節點開始走

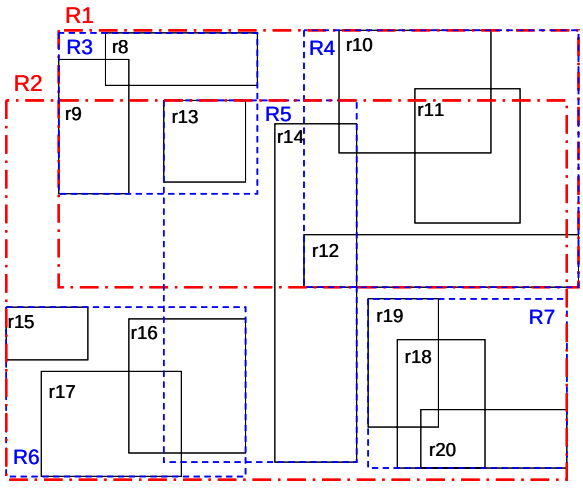


圖 2(a). 建築設施的空間物件分佈圖。

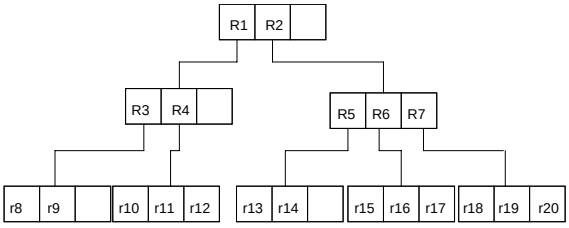


圖 2(b). 建築設施資料庫的 R-tree 架構圖。

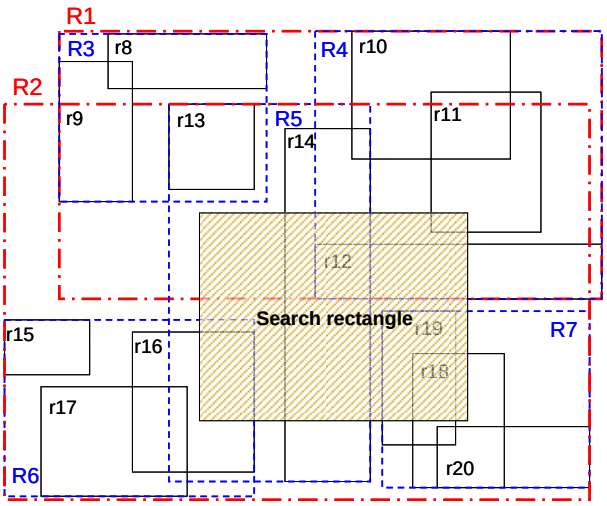


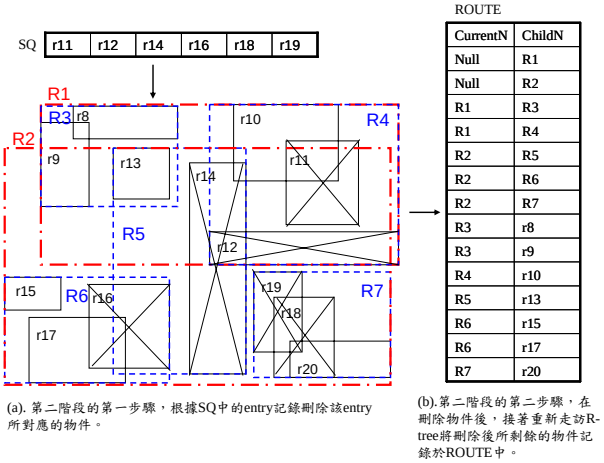
圖 3. 建築設施與工業區(search window)之間的重疊圖。

訪，找出與 SW 有重疊關係的物件並將它們儲存於 SQ 中。圖 4 為判斷重疊走訪過程中每一階層的 SQ 內容。在這之後，我們取出 SQ 中的所有

Level 1 - SQ	R1	R2				
Level 2 - SQ	R4	R5	R6	R7		
Level 3 - SQ	r11	r12	r14	r16	r18	r19

圖 4. 判斷重疊過程中每一階層的 SQ 內容。

記錄，分別為r11、r12、r14、r16、r18 和r19 並將它們從R-tree中刪除。在刪除物件後走訪R-tree將所有剩餘物件的路徑相關Spread資料存入ROUTE，其結果圖 5 所示。現在我們可以得知R_{obj}的值為 7，分別為r8、r9、r10、r13、r15、r17 和 r20 而 N_{cap} 的值為 3。因此我們需要 $N=\lceil R_{obj}/N_{cap}\rceil=\lceil 7/3\rceil=3$ 個葉節點來容納所有物件，但必須先將剩餘物件分成 3 群。依據[4]的作法，首先，從物件中選出相距最遠的 3 個物件做為各葉節點的第一筆記錄。因此，物件r8、r17 和 r20 分別為 3 個群組的第一個物件。接著以MBR最小擴張範圍為原則來分配剩餘的物件，使每個葉節點容量達半滿以上。在完成物件分配之後，接著向上調整節點的MBR直到根節點為止。圖 6(a)和圖 6(b)為R-tree完成多物件刪除後的結果。



(a). 第二階段的第一步驟，根據SQ中的entry記錄刪除該entry所對應的物件。

(b).第二階段的第二步驟，在刪除物件後，接著重新走訪R-tree將刪除後所剩餘的物件記錄於ROUTE中。

圖 5. 重新記錄刪除物件後 R-tree 所剩餘的物件。

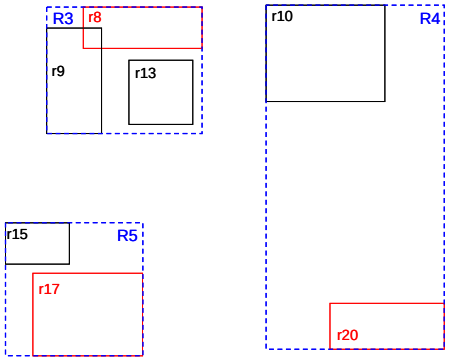


圖 6(a). 完成刪除動作後的物件分佈圖。

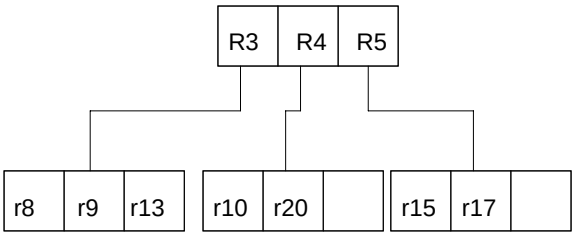


圖 6(b). 完成刪除動作後的 R-tree 架構圖。

5.結論

對於資料庫資料的操作，包括基本的搜尋、新增、刪除、修改等，傳統 R-tree 的使用都只針對單一物件的處理，少有同時對於多個物件進行處理，但在實際的應用中，使用者也可能需要一

次處理大量物件的情況，所以對多個物件進行操作是有必要的。R-tree 在物件刪除後可能有節點處於不平衡狀態，我們採用合併節點的方法來取代傳統物件重新插入的方式，理由如下。因為當 R-tree 刪除物件後，不平衡節點的剩餘物件，若使用傳統重新插入的方式，會因重新插入的次數過多而不斷地調整 R-tree 結構導致系統績效降低。我們的作法是一次刪除所有欲刪除的物件，計算容納剩餘物件所需的葉節點，將所有物件全部重新分配，讓多個物件刪除的過程中只需進行一次的向上調整，同時也避免了 R-tree 在刪除物件後高度變小，又因物件重新插入而使得高度變大。傳統 R-tree 重新插入的方式，對 R-tree 結構的調整只能達到區域最佳化(local optimization)，而重新分配的方式則可達到全域最佳化(global optimization)的結果。

參考文獻

[1] N. Beckmann, H.P., Kriegel, R. Schneider, B. Seeger, "***The R*-tree: An Efficient and Robust Access Method for Points and Rectangles***," Proc. ACM SIGMOD Int. Conf. on Management of Data, Atlantic City, NJ, pp.322-331, 1990.

[2] T. Brinkhoff, H.P., Kriegel and B. Seeger, "***Efficient processing of spatial joins using R-trees***," in: Proc. ACM SIGMOD Int. Conf. on Management of Data, pp.237-246, 1993.

[3] V. Gaede and O. Gunther, "Multidimensional access methods," ACM Computing Surveys, pp.170–231, 1997.

[4] A. Guttman, "***R-trees: a dynamic index structure for spatial searching***," in: Proceedings of the ACM SIGMOD, pp.47-57, 1984.

[5] P.W. Huang, P.L. Lin, H.Y. Lin, "***Optimizing storage utilization in R-tree dynamic index structure for spatial databases***," Journal of Systems and Software of Elsevier Science, 55(3), pp.291-299, 2001.

[6] A. Kumar, "***G-tree: A new data structure for organizing multidimensional data***," IEEE Trans. Knowl. Data Eng. 6(2), pp.341–347, 1994.

[7] J. Nievergelt, H. Hinterberger and K.C. Sevcik, "***The grid file: An adaptable, symmetric multikey file structure***," ACM Trans. Database Syst. 9(1), pp.38–71, 1984.

[8] J.M. Patel, and D.J. DeWitt, "***Partition Based Spatial-Merge Join***," in: Proc. ACM SIGMOD Int. Conf. on Mangement of Data, pp.259-270, 1996.

[9] J.T. Robinson, "***The K-D-B-tree: A search structure for large multidimensional dynamic indexes***," In Proceedings of the ACM SIGMOD International Conference on Management of Data, pp.10–18, 1981.

[10] N. Roussopoulos, D. Leifker, "***Direct spatial search on pictorial databases using packed R-trees***," In: Proceedings of the ACM SIGMOD, pp.17-31, 1985.

[11] T.Sellis, N. Roussopoulos, C. Faloutsos, "***The R+-Tree: A Dynamic Index for Multi-Dimensional Objects***," Proc. 13th Int. Conf. on Very Large Databases, Brighton, England, pp.507-518, 1987.

[12] K. Sevcik and D N. Koudas, "***Filter trees for managing spatial data over a range of size granularities***," In Proceedings of the 22th International Conference on Very Large Data Bases (Bombay), pp.16–27, 1996.