支援具有複雜關鍵字與結構限制之 XML 查詢系統

Supporting XML Query Processing with Complex Keyword Constraints

吳政儀　羅誠正　張雅惠

國立臺灣海洋大學 資訊工程研究所

{M94570060, Henry.lo, yahui}@ntou.edu.tw

摘要

隨著 XML 文件的廣泛使用與日益增加，如何從中查詢出符合使用者需求之結果，已經成為重要的議題。在本論文中，我們探討如何針對 XML 文件，找出符合使用者查詢句結構和資訊檢索限制之資料，後者包含對關鍵字距離和順序限制的處理。我們提出兩種作法，並評估其效益。第一種 TJ_IR 方法主要是先抓取符合節點標籤限制的元素，接著處理資訊檢索限制，最後處理整體結構的限制。第二種 SCU_Twig 方法，則是先抓取符合個別關鍵字限制的元素，接著找出這些元素的祖先中，何者符合資訊檢索的限制，最後再做路徑和整體結構的處理。我們會進行一系列的實驗比較此二系統在不同查詢條件下的效率。

Abstract

In this paper, we discuss how to retrieve the data satisfying the structural constraints and the full-text constraints imposed by users. We propose two methods. One is TJ_IR and the other is SCU_Twig. The first one first retrieves the elements corresponding to each tag constraints and then processes the keyword constraints. Then, the system uses the technique of TJFast to merge and feedback the final results. The second one will first retrieve the elements which match the keyword constraints, calculate the LCAs, and determine if they match the full-text constraints. Finally, the system will merge and feedback the results through the technique of TJFast. We have performed a series of experiments and discuss the performance of the two systems.

關鍵詞：XML、資訊檢索

一、序論

隨著XML文件的廣泛使用與日益增加，如何從中查詢出符合使用者需求之結果，已經成為主要的議題。為此，W3C制定了XQuery查詢語言 ， 允許針對XML文件的結構特性進行查詢 。但是由於XML也具有文件的特性，所以很多研究者也討論如何提供文件檢索 (或稱資訊檢索) 的功能，W3C也在其XQuery的Use Case中提出相關語法和範例。

因此，本論文是希望能夠針對XQuery中所討論的資訊檢索語法 ， 提供有效率的查詢處理。其中包含對結構限制的處理，以及對關鍵字距離和順序限制的處理。本論文提出兩種作法。第一種TJ_IR方法是將使用者輸入之查詢句結構建立成查詢樹，根據該查詢樹抓取符合節點的元素，接著處理關鍵字限制，再將符合的結果利用TJFast 的演算法[7] 合併。第二種SCU_Twig方法則是先透過SCU[1]的作法，抓取符合關鍵字限制的元素資料，再加入路徑的判斷，最後再套用TJFast的方式合併。實驗結果顯示，若是在處理複雜的資訊檢索限制時，TJ_IR系統優於SCU_Twig系統，不過就一般而言，SCU_Twig系統優於TJ_IR系統。

本論文的主要貢獻，總結如下所示：

1. 本論文首先提出TJ_IR方法，該方法擴充TJFast的作法，讓使用者在取得符合結構限制的XML資料上，可以同時針對資訊檢索的限制做進一步的處置。
2. 本論文另外提出SCU_Twig方法，其方法擴

充SCU的作法，使其在取出符合資訊檢索的限制上，並能對查詢句的結構限制做處理。

3.我們針對此兩種方法進行一系列的實驗，得知在有複雜關鍵字限制的情況下適合使用TJ_IR方法，但就一般而言SCU_Twig方法優於TJ_IR方法。

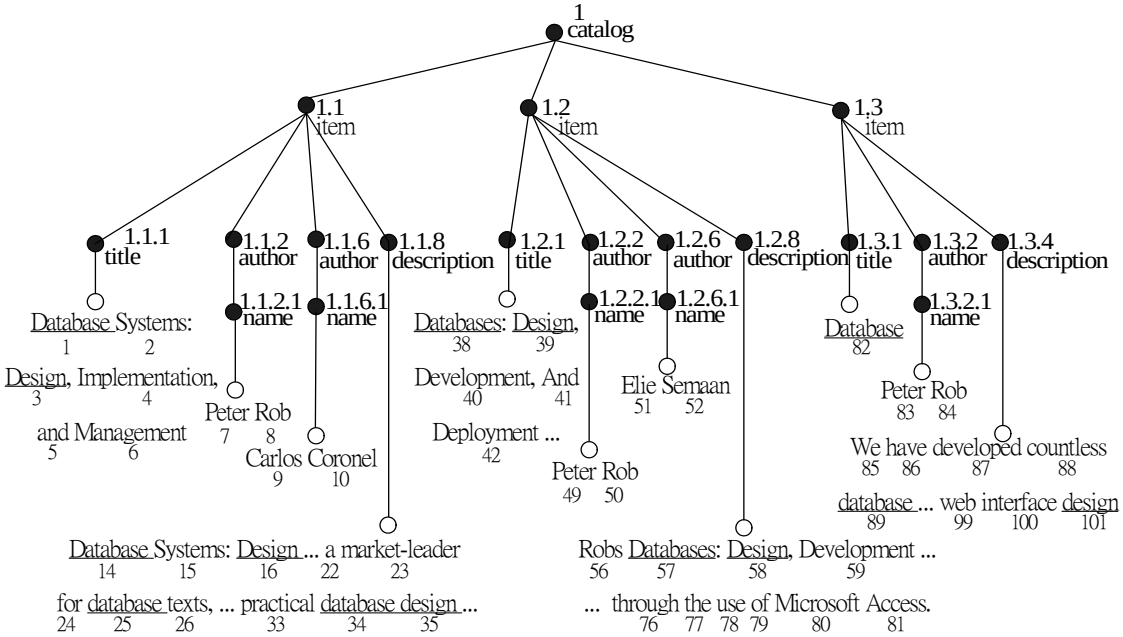
本論文其餘各節的架構如下：在第二節我們說明本論文所欲解決的問題和所提出的作法。在第三節中我們介紹資料的表示方式。在第四節中我們介紹TJ_IR的整體系統架構，並說明我們如何擴充其功能及所使用之演算法。在第五節我們將說明SCU_Twig的整體系統架構，並舉例說明其處理流程。在第六節中，我們將以實驗來分析兩系統的效率。第七節討論相關研究，最後在第八節我們將提出結論，並指出此論文未來的研究方向。

二、問題描述

我們將簡介在此論文中所需要用到的相關定義，如：XML、XQuery 等，並對本論文所解決的問題進行定義。

2.1 XML 樹

首先，我們介紹 XML 文件的基本結構。XML 文件以元素作為資料表示的基本單位，如 <name> Chang </name> 表示了一個 “name” 元素，其內容為 “Chang” ，而元素間必須具有嚴謹的巢狀包含關係，如 <author><name> Chang </name> <name> Liu </name></author> 表示了 author 元素中包含了兩個 name 元素。所以，我們通常將一份 XML 文件表示成一棵樹，在圖一的 XML 樹中，catalog 代表了文件的根元素，其下的三個 item 子元素，分別表示了三本書的資料，包含其書名 (title)、作者名稱 (author//name) 和書本內容的描述 (description)。其中，實心圓表示「元素節點」，旁邊的文字是標籤名稱，例如 item 是一個標籤名稱。空心圓表示「內容節點」，其下的文字是元素內容。直線則表示元素間的巢狀關係。另外，我們稱每條路徑最深之節點但不包含內容節點為「樹葉節點」 (Leaf Node) ，如 title。



圖一　XML 樹

```
L01 For $p in document
L02 ("http://dblab.cs.ntou.edu.tw/book.xml")
L03             /catalog/item
L04 Where $p/description ftcontains ("database" and
L05 "design" ordered) ftand("database" and "design" with
L06    distance at least 2 words) and $p//name ftcontains
L07 ("Peter" and "Rob" ordered)
L08 Return   $p
```

圖二：XQuery Full-Text 語法之範例

2.2 XQuery 語法

在本節中，我們介紹在 W3C 定義的 XQuery 的 Use Case 中，如何針對 XML 文件下達具有結構和資訊檢索限制之查詢句。在圖二的範例中，for 子句讓變數可以遞迴取得一個運算式的結果、where 子句允許對變數做條件的限制、而 return 子句則可以建構新的 XML 元素作為查詢的輸出。特別注意的是，在 L0-L07 的 Where 子句中，利用 ftcontains 函數列舉利用關鍵字進行全文檢索的三個限制式，分別用小括弧括起來。其中，第一個限制式，利用"ordered"的語法，要求"database"出現在"design"之前。至於第二個限制式，則是利用 "with distance" 語法，來規範關鍵字之間的距離
，在此我們要求"database"和 "design"的距離至少 2 個字。此二限制式都是針對 description 元素的內容。至於 L06 行，則是對另一條路徑下達 ordered 的限制 。

為了清楚地表達查詢句中的結構和關鍵字限制，我們將其表示成查詢樹。對應圖二的查詢樹如圖三所示。針對結構的部分，實線的圓形代表元素節點，粗實線的圓形表示該節點要回傳，如 item，而圓形之間的 "|" 代表父子關係， "||" 代表祖孫關係。另外，虛線矩形為內容節點，分別表示"distance"和"ordered"的限制。其中，每一筆 distance 限制式以 (term1, term2 ,operator, number) 表示，term 代表的是關鍵字，operator 代表的是兩個關鍵字之間距離的關係，如">"、"<"或"="，number 代表的是距離差距的限制。另一方面，每筆 order 限制資料以 [term1, term2] 方式標示，其意義為 term1 必須出現在 term2 之前。

2.3 相關定義

本篇論文，希望針對 XQuery 之結構查詢與資訊檢索限制，提出有效率的作法，為了方便之後的說明，首先我們提出以下定義。

[定義 1]：T_match：若 XML 樹中某一節點的標籤符合查詢樹中的標籤限制，則稱該節點為一個 T_match。

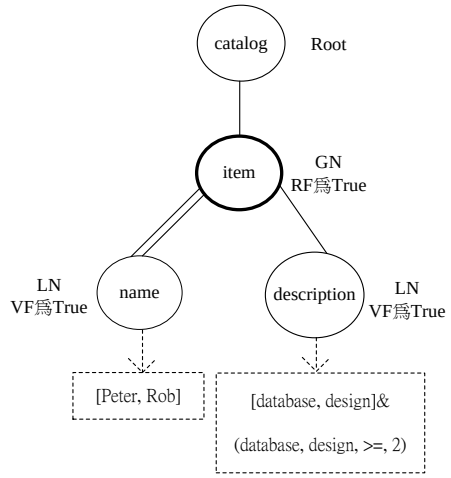
譬如，圖一中的節點 1.1、1.2 和 1.3，分別符合圖三查詢樹的標籤限制 "item"。

[定義 2]：P_match：若 XML 樹中某一個節點其對應的標籤路徑（labeled path），符合查詢樹中 root 到某個節點的路徑限制，則稱該節點為一個 P_match。

譬如，圖一中的節點 1.1.2.1，其標籤路徑為 "/catalog/item/author/name"，對應到查詢樹中的 "/catalog/item//name"。

[定義 3]：K_match：若 XML 樹中某一節點的內容存在查詢樹中要求的關鍵字，則稱該節點為一個 K_match。

譬如，圖一中的節點 1.1.1、1.1.8、1.2.1 、1.2.8、1.3.1 和 1.3.4 的內容分別包含圖三查詢樹中要求的關鍵字 "database"。我們在此遵循傳統資訊檢索 stemming 的作法，不考慮 "s"、"ing" 等之區別。



圖三： 查詢樹範例 (對應到圖二的 XQuery)

[定義 4]：C_match：若 XML 樹中某節點的內容符合查詢樹中相對應關鍵字的所有 ftcontains 限制，則稱該節點為一個 C_match。

譬如，圖一中的節點 1.1.8 和 1.3.8 符合圖三查詢樹中 description 的兩個 ftcontains 限制。1.2.1 和 1.2.8 之所以不符合，是因為 database 和 design 的距離只有 1。

[定義 5]：match tree：若在 XML 樹中的一群節點，每一個點皆為 P_match 或 C_match，且整體符合查詢樹的結構限制，則這群節點形成的子樹為 match tree。

譬如，在圖一中的節點集合 {1、1.1、1.1.2.1、1.1.8} 和 {1、1.3、1.3.2.1、1.3.4} 為符合圖三查詢句之 match tree。

[定義 6]：答案 (answer)：在 match tree 中符合 return 路徑的元素。

延續上例，節點 1.1 和 1.3 符合 return 路徑，所以為一個答案。

本論文提出兩個作法，分別擴充 TJFAST 和 SCU 的作法，以便能完整地處理查詢樹的結構和資訊檢索限制。第一個作法是以 TJFAST 的方式為主，所以稱為 TJ_IR。該作法會先找出 T_match，接著先處理 ftcontains 的限制，從中篩選出 C_match 之後，再利用 TJFAST 的方法找出 P_match 並組成 match tree。至於第二個作法，在找出 K_match 之後，我們會找出其祖先進行 C_match 的篩選。不過由於篩選出來的祖先不見得符合查詢樹路徑的限制，所以我們會再利用 TJFast 的方法找出 P_match 並組成 match tree。此作法我們稱為 SCU_Twig。

我們將描述此兩種作法的相關演算法，並進行實驗比較其效率。

三、資料表示

在本節中，我們介紹本論文針對 XML 資料的表示法，包含「延伸杜威編碼」、「元素編碼表」和「關鍵字編碼表」。

3.1 延伸杜威編碼

在本論文中，我們採用 TJFAST 提出的延伸杜威編碼 (Extended Dewey Code) ，其好處是可以直接將該編碼轉換成樹根到樹葉的標籤路徑 (Labeled path) ，譬如圖一中的 1.1.1 會轉換成/catalog/item/title。延伸杜威編碼的編碼公式

，會利用 DTD 的資料，完整的編碼公式請參見原論文[7]。

3.2 TJ_IR 的元素編碼表

在本節中，我們介紹 TJ_IR 系統如何表示 XML 資料。我們給予每個元素五項資訊，並將其以標籤分別儲存，如表一為標籤 title 對應的元素編碼表。前三項資訊包含 Extended Dewey Code、Level、Tagname，是屬於結構的資料，後兩項則為了進行全文檢索的處理，其中，position 是該關鍵字在 XML 文件中出現的位置。注意到，我們會針對文數值內容做 stopword 處理，把 stopword 剔除掉，以免關鍵字資料太過於龐大。另外，為了允許使用者對非樹葉節點的元素，直接表示資訊檢索的限制式，我們會紀錄一個元素之下所有的關鍵字，包括直接包含和間接包含的。

另外，我們會為元素編碼表建立一個索引，利用 tagname+level 為 key，以快速地取出某個 tagname 在 XML 文件中某特定 level 的資料。

表一：圖一中 title 的元素編碼表

Extended Dewey Code	Level	Tagname	Keyword	Position
1.1.1	3	title	Database	1
1.1.1	3	title	Systems	2
1.1.1	3	title	Design	3
1.1.1	3	title	Implementation	4
1.1.1	3	title	Management	6
1.2.1	3	title	Database	38
1.2.1	3	title	Design	39
1.2.1	3	title	Development	40
1.2.1	3	title	Deployment	42
1.3.1	3	title	Database	82

3.3 SCU_Twig 的關鍵字編碼表

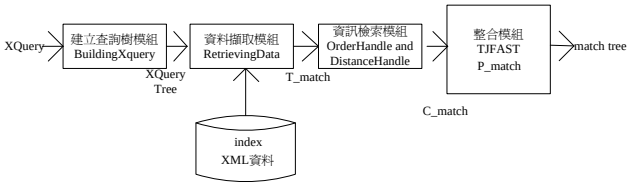
在本節中，我們介紹在 SCU_Twig 系統中，如何對 XML 文件關鍵字編碼。與 TJ_IR 不同的是，我們只給予每個元素三項資訊，包含延伸杜威編碼 (Extended Dewey Code) ，關鍵字 (Keyword) 和位置 (Position)，第一項是屬於結構的資料，後兩項則為為了進行全文檢索的處理。不記錄標籤 (tagname) 和深度 (level) 的原因，是因為利用關鍵字和位置資料處理完 K_match 和 C_match 後，我們會直接利用延伸杜威編碼產生 P_match，所以不需要標籤資訊。

表二：圖一中 database 的關鍵字編碼表

DeweyID	global_position
1.1.1	1.1.1.1
1.1.8	1.1.8.14
1.1.8	1.1.8.25
1.1.8	1.1.8.34
1.2.1	1.2.1.38
1.2.8	1.2.8.57
1.3.1	1.3.1.82
1.3.4	1.3.4.89

四、TJ_IR 系統

在本節中，我們介紹 TJ_IR 系統的作法，整體架構如圖四所示。首先，當使用者輸入 XQuery 查詢句，我們會將其建立成查詢樹。接著，資料擷取模組會根據該查詢樹抓取符合個別標籤的元素，亦即 T_match。為了能夠快速的利用元素的標籤取出對應的 XML 元素，我們會事先建立 index。若該節點有關鍵字的限制，則由資訊檢索模組，分別處理 order 的限制和 distance 的限制，以得到 C_match。最後，在整合模組中，會將之前取出的元素組合，以形成符合整個查詢句結構之答案。在下面的小節中，我們將說明其中比較主要的三個模組。

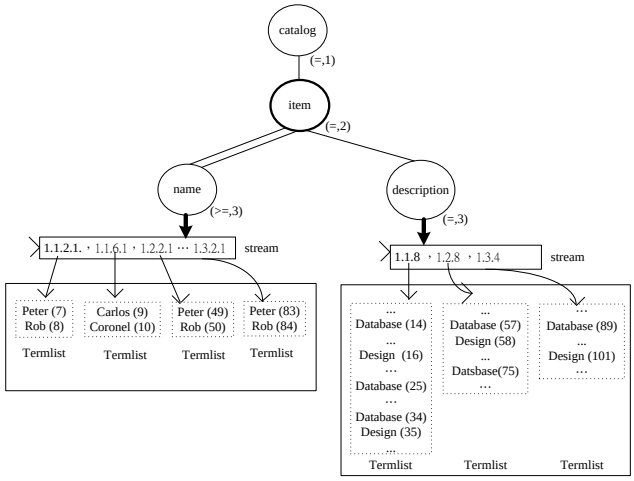


圖四：TJ_IR 之系統架構圖

4.1 資料擷取模組

此模組會使用在第三節中所介紹的資料結構配合上查詢樹，將所需要處理的資料取出。

此模組的輸出如圖五所示，基本上延續圖三的查詢樹結構，但是利用到兩個額外的資料結構，stream 和 Termlist。其中，查詢樹的葉節點會對應到一個 stream，記錄符合該節點標籤的元素編碼，至於每一個元素其內表示的所有關鍵字及位置，則記錄在 Termlist 中。圖五的圖對應到圖二的查詢樹和圖一的 XML 資料。



圖五：查詢樹取得初步資料的範例

4.2 資訊檢索模組

接下來說明如何處理資訊檢索的限制。該模組會處理 stream 中的每個元素，看其下的關鍵字能否符合資訊檢索的限制，圖六為處理 Order 限制之演算法。OrderConstraint 記錄了一筆 ordered 的限制式。該程式會先到 TermList 尋找是否出現 OrderConstraint 中的第一個關鍵字

，若該關鍵字存在，再接著尋找第二個關鍵字。譬如以圖二的 [database,design] 和圖五為例，會先找出 TermList 中的 database，接著再向下找 design。

```
演算法名稱：OrderHandle
輸　　入：TermList, OrderConstraint
輸　　出：True/False
L01 If OrderConstraint==NULL return TRUE;
L02 Term1Match = FALSE; /*flag*/
L03 Term2Match = FALSE; /*flag*/
L04 while (TermList != null) {
L05    if (TermList->keyword == OrderConstraint->Term1)
L06       and (not Term1Match)
L07    {
L08       Term1Match = TRUE;
L09    }
L10    if (Term1Match) and (TermList->keyword ==
L11 OrderConstaint->Term2)
L12    {
L13       Term2Match = TRUE;
L14       break;
L15    }
L16    TermList = TermList->next;
L17 }
L18 if (not Term2Match)
L19    return FALSE;
L20 else
L21    return TRUE;
```

圖六：處理 ordered 限制的演算法

圖七為處理 distance 限制之演算法。其基本精神是，我們首先利用第一個關鍵字出現的位置，以及在限制式裡限定的距離，找出一個範圍，以 LeftBoundary 和 RightBoundary 表示。若 distance 限制式要求的是 “<”或“＝”的限制，則第二個關鍵字必須出現在該範圍內。若 distance 限制式要求的是 “>”的限制，則第二個關鍵字不能出現在該範圍內。如此可以縮小比對的範圍。以下作進一步的描述。

L01 行到 L03 行將指標移動到第一個關鍵字 (T1) 和第二個關限字 (T2) 在 Termlist 中第一個出現的位置，假使 T1 或 T2 有一個不存在，則在 L04 行到 L05 行回傳 false。L06 行到 L37 行為處理 distance 的主要演算法。因為論文空間的關係，在此只列出處理 “<” 和 “ >” 的程式碼。

```
演算法名稱：DistanceHandle
輸　　入：TermList, DC
/* TermList 為紀錄 Term 資訊的 list */
輸　　出：True/False
變數說明：
T1Cursor, T2Cursor/*對應 T1 和 T2 在 Termlist 中的指標*/
T1Position, T2Position /*兩個關鍵字的位置*/
LeftBoundary, RightBoundary /*記錄第二個關鍵字間可能的範圍*/
L01 T1Cursor = T2Cursor = TermList;
L02 T1Position = GetPosition(T1Cursor, DC->T1);
L03 T2Position = GetPosition(T2Cursor, DC->T2);
L04 If (T1Position==0 or T2Position==0)
L05    return False;     /*有一個 term 沒找到*/
L06 switch(DC->operator){
L07   case"<":
L08      while (T1Position !=0 and T2Position !=0)
L09                /*兩個關鍵字都存在*/
L10       {      /*計算 t2 可以出現的範圍*/
L11         LeftBoundary = T1Position - DC->number;
L12         RightBoundary = T1Position + DC->number;
L13       if (T2Position < LeftBoundary){
L14                  /*假使 t2 太前面*/
L15          T2Position=GetPosition/*往下找 t2 的下一個
L16 位置*/(T2Cursor,DC->T2);
L17          continue;                  /*繼續比較*/
L18        }elseif (T2position > RightBoundary)
L19                /*t2 超過右邊的範圍*/
L20        {   /*往下找 t1 的下一個位置*/
L21          T1Position=GetPosition (T1Cursor, DC->T1);
L22          continue;                  /*繼續比較*/
L23        }else
L24          return true;
L25     }
L26   case ">":
L27     T1lastPosition=GetlastPosition(T1Cursor, DC->T1);
L29     T2lastPosition=GetlastPosition(T2Cursor, DC->T2);
L30     if(T1Position !=0 and T2Position !=0) /*兩個關鍵字
L31 都存在*/ {
L32       if(T2Position<T1Position-DC->number or
L33 T2LastPosition>T1Position+DC->number or
L34 T2Position<T1LastPosition-DC->number or
L35 T2LastPosition>T1LastPosition+DC->number)
L36          return true;
L37       }
L38 }return False;
```

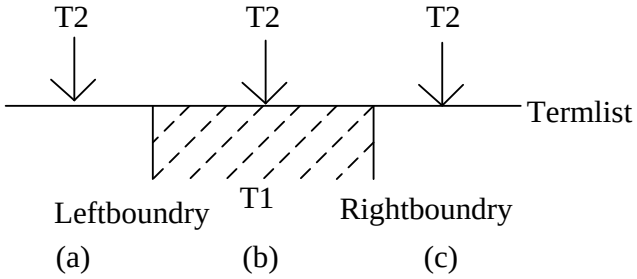
圖七：處理 distance 限制的演算法

L07 行到 L25 行為處理“<”的情況，L11 行到 L12 行首先計算出 T2 可以出現的範圍。假設第一個關鍵字 database 的位置為 14，而查詢句要求差距在 7 以內，經由運算 [14-7, 14+7] 形成的範圍為 [7, 21]。L13 行到 L17 行對應到圖八 (a)，表示目前 T2 的位置太前面，則必須移動 T2Position 再繼續判斷。L18 行到 L22 行對應到圖八 (c)，表示 T2 的位置已經超過目前應該出現的範圍，位置太後面，所以移動

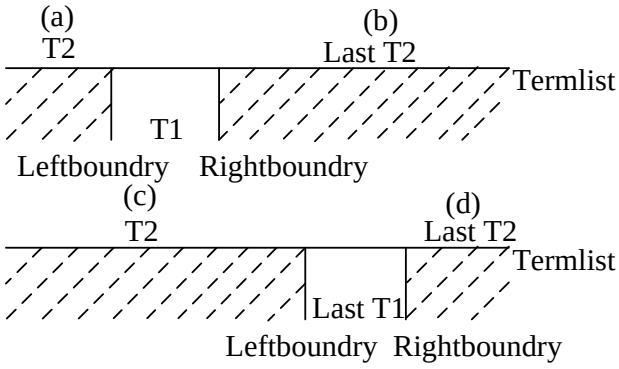
T1Position 到下一筆資料，繼續處理。若滿足範圍如圖八 (b) 則在 L24 行回傳 true。

L26 行到 L37 行則是處理“>”的情況，由於兩者的距離越遠越好，所以我們只針對 T1 的第一個位置和最後的位置、T2 的第一個位置和最後的位置共 4 種情況做處理，如圖九 (a) (b) (c) (d) 所示，符合限制則為 True。

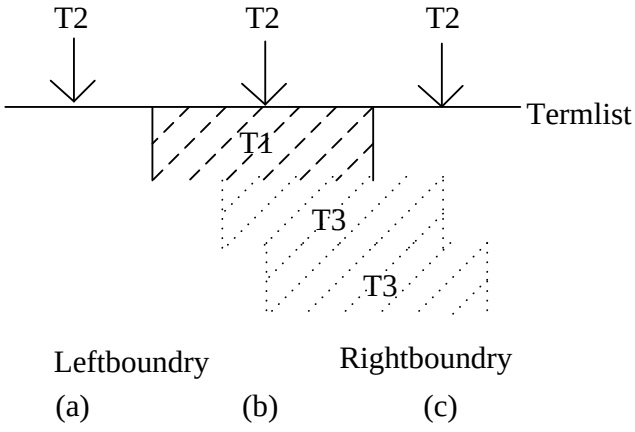
以下我們說明處理“=”的情況，首先同樣利用 T1 計算出 T2 的兩個邊界，但是在此情況中 T2 只有恰好在 LeftBoundary 和 RightBoundary 上才是符合結果。我們先判斷 T2Position 是否符合，若是則回傳 true，否則判斷該移動 T1Position 或 T2Position，這裡分成三種可能。前兩種可能和“<”的移動方式類似 (參見圖十的(a)和(c)) ，第三種種可能是 T2 恰好在範圍內如圖十 (b) 所示，則需再度判斷 T2 是否會和此範圍內的其他關鍵字符合距離限制，首先我們將 T1 指標指的資料暫時丟給 T3 指標使用，並取得 T3Position 的下一筆資料，接著判斷 T2 的位置是否太前面，若是則持續移動 T3 的指標看能否與 T3 的邊界相等，若是則回傳 true。反之則持續移動 T2 的指標看能否與 T1 的邊界相等，否則繼續處理下一筆 T1 直到結束。



圖八：處理 distance 限制範例 “<”



圖九：處理 distance 限制範例 “>”

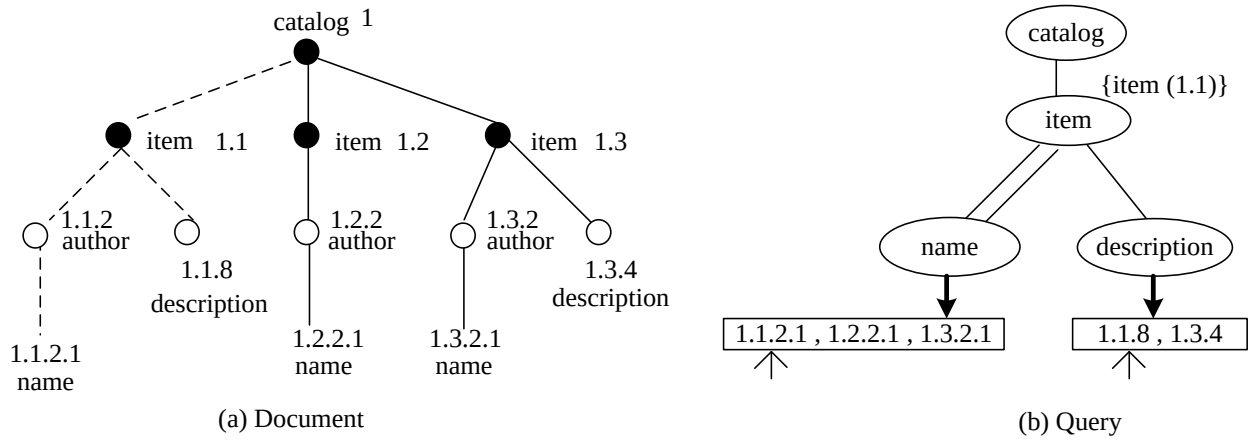


圖十：處理 distance 限制範例 “=”

4.3 整合模組

最後，在整合模組中，將針對資訊檢索模組處理過後的結果的整合。首先會針對留在 stream 中的元素，利用延伸杜威編碼的特性依序將其還原成標籤路徑，並同時和查詢樹中對應的路徑作判斷，以取得 P_match，最後再利用 TJFAST 的合併方式將其結合，輸出最後符合結構和關鍵字限制的答案。

如圖十一為 TJFAST 的範例，(a)圖為 XML 樹，而(b)圖為查詢樹針對查詢句的兩條路徑 (/catalog/item//name 和 /catalog/item/description)

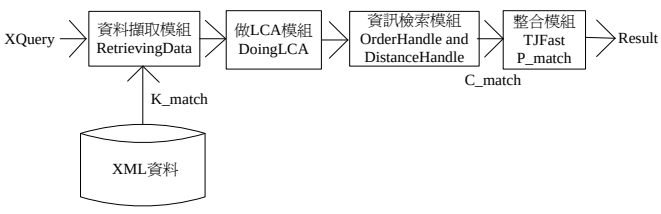


圖十一：整合模組的範例

的 stream，該演算法依序判斷指標指到的元素，在 XML 樹中是否有符合分枝結構的元素。在此，兩個 stream 中的 1.1.2.1 和 1.1.8 在 XML 樹中對應到元素 1.1，符合分枝元素 item 的限制，所以找出第一筆符合結構和路徑限制的答案。

五、SCU_Twig 系統

在本節中，我們介紹 SCU_Twig 的作法，如圖十二所示為整體的系統架構圖。首先，當使用者輸入 XQuery 查詢句，資料擷取模組根據該查詢句抓取符合個別關鍵字的元素，即 K_match，為了能夠快速的找到 XML 資料中的元素，我們會利用 inverted list 的技巧，以利用關鍵字取出對應的節點資訊。接著我們針對所有取出來的元素做 LCA (Lowest Common Ancestor)，取出節點間最小的共同祖先。若該節點有關鍵字的限制，則由資訊檢索處理模組，分別處理 order 的限制和 distance 的限制，以得到 C_match。最後，再整合模組中，會透過 TJFast 的方式合併，以形成符合整個查詢句結構之答案。



圖十二：SCU_Twig 之系統架構圖

5.1 SCU Table

首先，我們說明 SCU (Smallest containing Unit) 表格的結構。每一組 SCU 表格由相同的關鍵字，但是不同的節點 (元素) 和對應位置串起組合而成。如圖十三所示，為取出以 database 為關鍵字的 SCU 表格範例，資料包含節點 N (node)、關鍵字 P (pattern) 和關鍵字的位置 M (match)。SCU 表格裏記錄的都是直接包含關鍵字的節點，此方式可以降低在作 LCA 時，處理祖先時間上多餘的花費。SCU Table 表格由多筆 item 組成，每一筆 item 包含杜威編碼和關鍵字，另外由於在做條件限制判斷時，會將子孫節點判斷後不符合限制的資料傳給祖先繼續處理，因此會將關鍵字資料串連起來。

N (node)	P (pattern)	M (match)
1.1.1	database	1.1.1.1
1.1.8	database	1.1.8.14
1.1.8	database	1.1.8.25
1.1.8	database	1.1.8.34
1.2.1	database	1.2.1.38
1.2.8	database	1.2.8.57
1.2.8	database	1.2.8.75
1.3.1	database	1.3.1.82
1.3.4	database	1.3.4.89

圖十三：SCU Table 範例

5.2 演算法說明

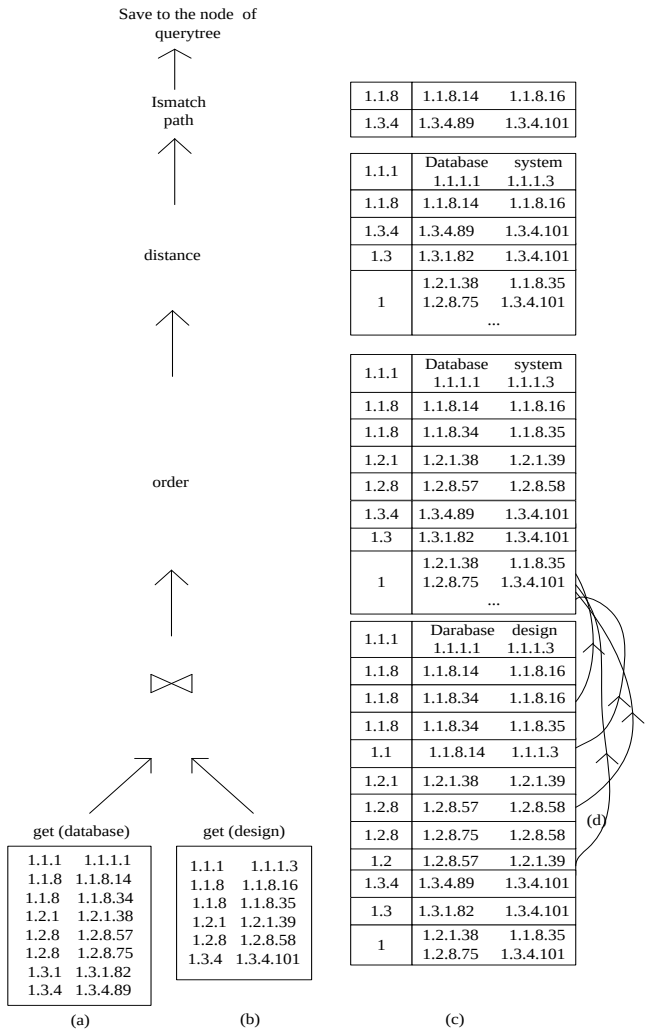
這一節說明個別演算法。圖十四為 SCU_Twig 之主程式，在 L01 行先對 XQuery 建查詢樹，在 L03 行到 L10 行利用 hash table 取出 where 限制式中的每個關鍵字對應的 SCU Table，接著再依序處理限制。在 L12 行到 L14 行將取出的關鍵字透過 LCA 的方式合併處理，在 L15 行到 L21 行，我們處理資訊檢索的限制，接著將符合的結果放入 XqueryTreeNode 的 stream 中。最後在 L23 行呼叫 TJFast 的演算法合併查詢樹，以輸出符合查詢限制的結果。

```
演算法名稱：SCU_Twig
輸入：XQuery /*XQuery 解析模組所解析出的路徑表*/
輸出：MTSet
變數：KO1,KO2 /*查詢句中需處理限制的第一個關鍵字和第二個關鍵字*/
L01 XQueryTree=BuildingXQueryTree(XQuery);
L02 For(each constraint in where_constraint){
L03     If KO1 in hashTable
L04         Get KO1.SCUTable using hashTable
L05     Else
L06         Get KO1.SCUTable from keywordfile
L07     If KO2 in hashTable
L08         Get KO2.SCUTable using hashTable
L09     Else
L10         Get KO2.SCUTable from keywordfile
L11     LCA.SCUTable = NULL;
L12     if(KO1.SCUTable and KO2.SCUTable 不為空)
L13         LCA. SCUTable=DoingLCA(KO1. SCUTable,
L14         KO2. SCUTable);
L15     if(LCA. SCUTable 不為空){
L16         Match_DeweyidList=
L17         Doing_predicate(LCA.SCUTable, constraint);
L18         stream=Ismatch(Match_DeweyidList, where 的路
L19         徑);
L20         T.stream=AddtoElement (stream);
L21         SavetoXqueryTreeNode(T.stream)}
L22     }
L23 }MTSet = TJFast(XQueryTree)
```

圖十四：SCU_Twig 之演算法

以下我們簡單說明，處理 LCA 的演算法，完整的演算法請參見論文[1]。該演算法首先針對兩個 SCUTable 給予各自的起始指標，計算出第一筆 LCA 資料，接著再根據兩個 Table 中杜威編碼的前序關係，線性移動指標，減少在做 LCA 時，時間上的浪費，最後再將結果輸出。例如圖十五所示，由 “database” 圖十五 (a) 和 “design” 圖十五 (b) 的 SCU 表格計算出 LCA 結果表示成圖十五(c)。為了方便表示，關鍵字 (P) 只表示在第一列，其餘的只表示關鍵字存在的位置 (M)。

至於處理資訊檢索限制之演算法 (Doing_predicate) 分成兩種處理方式。一種為直接處理限制，若是符合限制則輸出。若是不符合資料的杜威編碼，則會將其關鍵字資料加入至其最近的祖先，再測試祖先，如此遞迴下去。例如十七 (c) 下圖所示，當 LCA (1.1) 不符合 order 限制時，則會將其加入至祖先 LCA (1) 的資料中，如箭頭十七 (d) 所示。



圖十五：查詢範例 (對應到圖二的 XQuery)

針對個別資訊檢索的限制處理，基本上我們直接取出對應個別關鍵字的位置，然後直接判斷。假設限制式是 order，我們直接判斷兩者間的順序是否符合限制。譬如以圖十五為例，database 必須在 system 之前，所以 database 的位置必須小於 system 的位置，即 database (1.1.1. "1") < design (1.1.1. "3") 符合限制。

至於 distance 限制之演算法，則區分成 "<" 、 "=" 和 ">"的處理。以圖十五 (c) 為例，若 database 和 design 的距離必須大於 1，則 LCA (1.1.1) 資料中的距離 |3-1|>1 符合限制，而 LCA (1.2.1) 資料中的距離 |38-39|>1 則不符合限制。

六、實驗

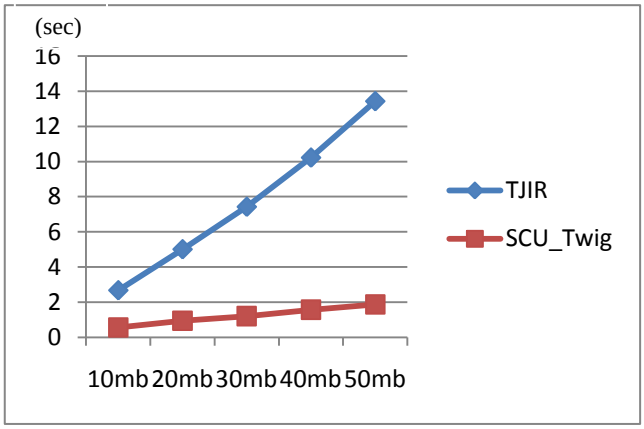
在本節中我們將設計數個實驗來評估我們所提出之兩個系統的效能。我們以個人電腦做為實驗的環境，其 CPU 為 Intel Core 2 1.9 GHz，其記憶體為 1.5GB，而採用的作業系統則為 Windows XP。

在此實驗中，我們將採用 XMark 之 Data Set，大小為 10MB 至 50MB，其文件最大深度為 12。

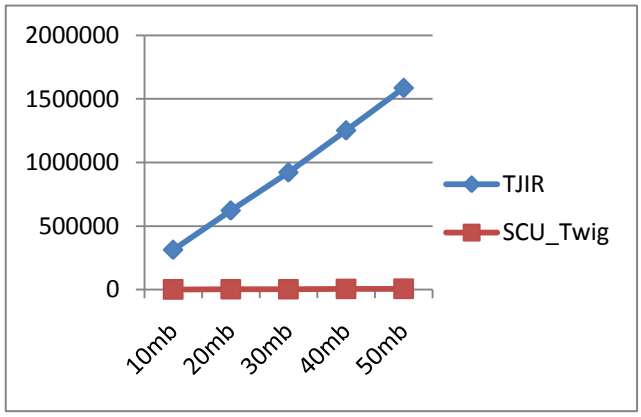
6.1 dataset 大小之實驗

在此實驗中，將針對查詢句在不同大小的 Data Set 的效能做比較。此查詢句 (Q1) 的資訊檢索限制只有簡單的 order 限制。

根據圖十六 (a)，TJ_IR 的效率明顯地比 SCU_Twig 差，而主要的原因是 TJ_IR 系統在抓取資料上比 SCU_Twig 系統所花費的時間還多，原因在於 TJ_IR 系統是針對符合標籤的節點 (T_match) 做處理，而 SCU_Twig 系統則是針對符合的關鍵字做取捨 (K_match) ，當 K_match 比 T_match 少時，所花費的時間就明顯較少，如圖十六 (b)。不過這兩個系統大致上都是呈線性成長的趨勢。



(a) 執行時間



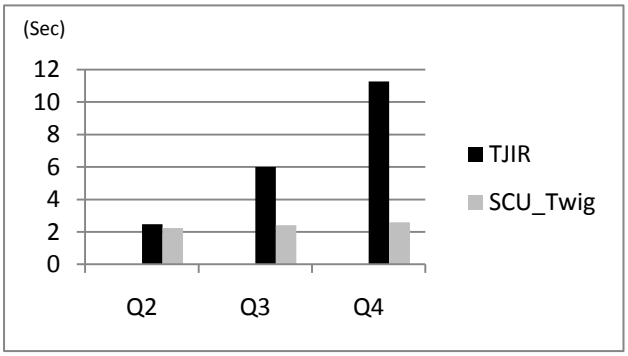
（b）資料節取模組之輸出數量

圖十六：Data Set 大小之實驗結果

6.2 不同節點限制之實驗

在本節中，我們給予複雜的資訊檢索限制，包含 order 和 distance，同時控制查詢樹葉節點的路徑長度 (深度) 。其中 Q2 的路徑最長，Q3 其次，而 Q4 的最短。圖十七所示為三個查詢句的執行時間。可以發現在 Q2 查詢句中，TJ_IR 系統的整體時間，比處理 SCU_Twig 系統的時略差而已，但是在 Q3、Q4 查詢句時卻

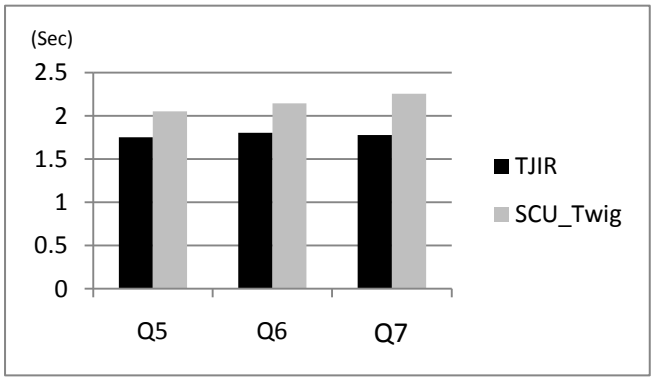
有明顯的差異，原因在於 TJ_IR 系統中，Q4 查詢句中的葉節點其對應的元素編碼表包含 Q2、Q3 關鍵字資訊，同理 Q3 查詢句包含 Q2 的資訊，使其在執行時所花的時間明顯地增加。



圖十七：不同節點限制之整體時間

6.3 相同節點不同距離限制之實驗

在此實驗中，我們控制查詢句 distance 限制中關鍵字之間的距離，Q5、Q6 和 Q7 分別為處理 distance > 5 words、distance > 50 words 和 distance > 500 words 之限制，如果距離愈大，則產生的 LCA 就會越高，也就是離 Root 愈近。圖十八顯示處理不同距離限制之整體執行時間。比較 TJ_IR 系統和 SCU_Twig 系統的執行結果，可以發現 SCU_Twig 會將不符合的結果傳遞給祖先繼續處理，因此造成在處理的時間上比 TJ_IR 系統多。



圖十八：相同節點不同限制之整體時間

七、相關研究

針對XML的查詢處理，首先討論的是如何處理XML的結構限制，有些論文針對twig pattern的狀況，設計串接的stack，以簡潔地表達符合結構的資料[3]。 論文[4] 進一步改善 [3] 的作法，提出了兩層式（外層包內層）的stack。該論文結合其Bottom-up的方法以及PathStack的Top-down的想法，在產生符合query的結果時立即列舉出來並清空Hierarchical stack內的data，以節省大量的記憶體空間。

由於XML資料具有文件的特性，所以IR的技術也被考慮。在研究 [2] 中，作者給與每個元素一個評分 (score) ，並在評分時同時考慮其下的子孫元素，作者提出了TIX (a bulk algebra) 及以stack為基本的TermJoin跟PhraseFinder演算法，TIX提供了將IR型式的查詢併入一般XML資料庫的可能，而以stack為基本的TermJoin及PhraseFinder能快速有效的將評分算出。至於在研究 [5] 中，作者將整個XML文件採用Dewey編碼並紀錄於inverted list中，如此一來在搜尋關鍵字時，可以由Dewey 編碼的前序排列得到符合的路徑，也可以快速地得知任兩個元素之間是否有父子或祖孫關係。

另一方面，研究 [6] 則提出將Structure Index 以及Inverted List結合的架構，其作法先將Branching Path Expression依分支的點將其分解成一條一條無分支的路徑，由Structure Index取出符合該結構的元素編碼，最後再透過Inverted List取得值符合的元素，並將其與之前所處理出的Structure比對看是否符合，若是則此為正確答案。論文[8] 中在XML的結構部分則是利用PreOrder和PostOrder對元素進行編碼，並將該編碼存放於Inverted List中。作者並修改基本的 top-K algorithm，使其能針對XML文件結構上的關係進行TOP K的處理。研究 [1] 則將

符合個別關鍵字的元素存放於SCU table中，然後在處理複雜的條件句，如限定不同關鍵字的順序，該論文會先找出符合所有關鍵字的LCA，接著再進行Full-Text Predicates的運算。

八、結論與未來方向

在本論文中，我們實作 TJ_IR 和 SCU_Twig 兩個系統，此兩系統利用 TJFAST 的合併方式以進行 XQuery 查詢處理，並且可以處理複雜的關鍵字限制。由實驗的結果得知，在 TJ_IR 系統中，主要的時間花費在取得資料上，時間多於 SCU_Twig 系統，但就其他模組而言，TJ_IR 系統也略遜於 SCU_Twig 系統。不過在處理複雜的 IR 限制上，TJ_IR 系統會優於 SCU_Twig 系統上的處理，因此在該情況下，適合使用 TJ_IR 系統，但就大部分的狀況，SCU_Twig 系統仍優於 TJ_IR 系統。

本論文未來的方向，可以針對 XML 資料不同的屬性事先做更適當的處理，以決定最佳的執行策略，另外也可以對查詢句中其他不同的限制，提出對應的處理演算法。

誌謝

此計畫由國科會贊助，編號為：

NSC97-2221-E-019-028

九、參考文獻

[1] Sihem Amer-Yahia, Emiran Curtmola, Alin Deutsch, "Flexible and Efficient XML Search with Complex Full-Text Predicates", In Proceeding of the SIGMOD Conference, Chicago, Illinois, USA, 2006.

[2] Shurug Al-Khalifa, Cong Yu, H. V. Jagadish, "Querying Structured Text in an XML Database", In Proceedings of the SIGMOD Conference, Jun. 2003.

[3] Nicolas B, Nick K, Divesh S. Holistic Twig joins, "Optimal XML pattern matching", In Proceedings of the SIGMOD Conference, 2002.

[4] Songting Chen, Hua-Gang Li, Junichi Tatemura, Wang-Pin Hsiung, Divyakant Agrawal, K. Selcuk Candan, "Twig2Stack: Bottom-up Processing of Generalized-Tree-Pattern Queries over XML Documents", In Proceedings of the VLDB Conference, Pages:283–294, September 12–15, 2007.

[5] Lin Guo, Feng Shao, Chavdar Botev, Jayavel Shanmugasundaram, "XRANK: Ranked Keyword Search over XML Documents", In Proceedings of the SIGMOD Conference, San Diego, CA, June 9-12, 2003.

[6] Raghav Kaushik, Rajasekar Krishnamurthy, Jeffrey F. Naughton, "Raghu Ramakrishnan: On the Integration of Structure Indexes and Inverted Lists", In Proceedings of the ICDE Conference, 2004.

[7] Jiaheng Lu,Tok Wang,Ling Chee-Yong Chan,Ting Chen, "From Region Encoding To Extended Dewey: On Efficient Processing of XML Twig Pattern Matching", In Proceedings of VLDB Conference, Pages: 193–204, Norway, 2005.

[8] Martin Theobald, Ralf Schenkel, Gerhard Weikum, "An Efficient and Versatile Query Engine for TopX Search", In Proceedings of the VLDB Conference, 2005.