數位典藏中後設資料之查詢處理設計

李政達　羅介璋　張雅惠
國立台灣海洋大學資訊工程研究所
202 基隆市北寧路 2 號
lct@cyber.cs.ntou.edu.twT　{M93570009 , yahui}@mail.ntou.edu.tw

摘要

數位典藏國家型計畫的目的，是希望將國家龐大的珍貴文物與檔案以數位化典藏，而為了便利資料的分享，文物的後設資料是以XML表示。因此，如何在大量的XML資料中，快速的選取出使用者所需要的資訊，已成為重要的研究議題。在本篇論文中，我們提出一個能處理基本的XQuery查詢句的演算法，首先將XQuery建立成查詢樹，並將其分解成若干個片段路徑，根據這些片段路徑取出部分資料後，再將這些部分資料接合成符合結構的完整資料。我們也執行數個實驗來證明我們作法有極佳的效率。

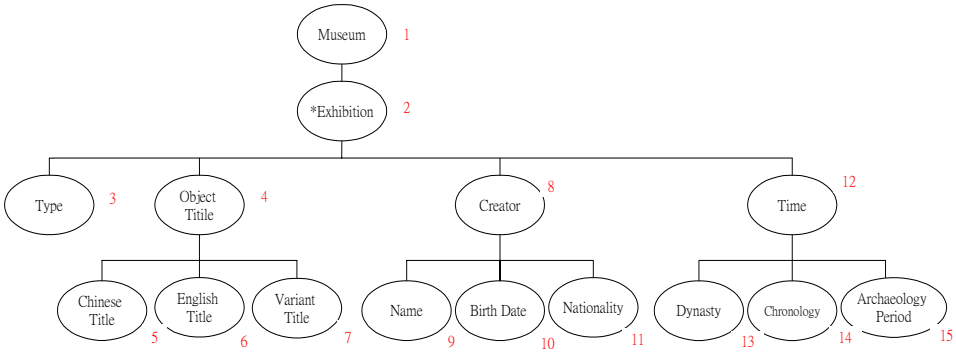
1、緒論

目前推動的數位典藏國家型計畫，企圖將國家龐大的珍貴文物與檔案以數位化典藏，其中，後設資料工作小組[1]，針對不同性質的文物、器皿及珍貴文史檔案等，訂定以 XML 描述的後設資料（Metadata），以便利資料的交換。因此，如何根據使用者的查詢句，從 XML 資料中有效率的取出符合使用者需求的資訊，便成為重要的研究議題。

近年來已有許多的研究從不同的觀點來解決 XML 查詢處理的問題，其中有的方法是先替大量的 XML 文件建立整合索引（structural summary），再透過該結構進行查詢。為了避免整合索引過大的問題，因此在研究[6]中提出了 A(k)-Index，主要根據元素間局部相似性建立 index，若兩個元素長度為 K 的 incoming path 相同則合併形成 index node，進而減少所產生的 index size。而在研究[7]與[8]中，則透過 work load 的方式紀錄使用者可能會輸入的查詢句，並根據 work load 所紀錄的查詢句中的節點，動態的建立 index 達到減少 index size 之目的。

另外有的論文，則針對 XML 查詢句中的路徑表示法處理加以研究。譬如，表示法“a//b”，是尋找在 a 元素底下所有的 b 元素，並沒有層數的限定，我們稱此為“跳層的”路徑表示法，而一般則將找出元素間結構上的關係稱作 structural join。在研究[2]中，作者將每個 XML 的節點以 preorder 及 postorder 編碼，透過這樣的編碼方式即可快速的決定任意兩個節點結構上的關係，也就是，若一個節點 A 為另一個節點 B 的 ancestor 或 parent，則 A.preorder < B.preorder 且 A.poster >B.postorder。但在 XML 中 A 節點可能會有多個 descendant 節點 B，相對的 B 節點也可能會有多個 ancestor 節點 A，所以在[2]中作者又利用 stack 來處理 structural join，以達到整體時間複雜度隨著 input 與 output 資料之總合成線性成長的效能。

研究[5]則建立額外的 Index 來找出符合特定結構的節點。他們提出 XR-Tree，其為一個以 B+ Tree 為基本的資料結構。當給與某個節點的 preorder 及 postorder 編碼時，XR-Tree 能快速的找出該節點所有的 ancestor 及 descendant 節點。研究[3]進一步將 XR-Tree 結合研究[2]中所提出的方式，來作 structural join，透過 XR-Tree 的特性，選取下一個需要進行 structural join 處理的節點，可跳過一些不



圖一：DTD 範例

需要處理的節點，如此即可減少需要作比較的節點，進而增進效率。

本篇論文針對能以樹狀表示的查詢句進行處理。首先，將查詢句中的所有路徑表示法用樹狀的方式表示以形成查詢樹，接著將查詢樹分解成若干個片段路徑，針對每個片段路徑從 XML 資料中取出符合該片段路徑的元素，稱之為中繼資料。如此切割的目的是便於我們取出資料。接下來，我們組合片段路徑形成片段接合路徑，組合的過程中，同時比對片段路徑的中繼資料是否能符合片段接合路徑，最後將這些資料結合成符合查詢樹的答案。整個系統的設計，類似於 “divide-and-conquer” 的精神。

本篇論文架構如下：在第 2 節，我們將介紹 DTD 、XML 和 XQuery 的表示法，在本篇論文中皆以樹狀的方式表示；於第 3 節中說明如何將查詢樹進行分解；於第 4 節說明如何組合將查詢樹分解後所產生的中繼資料；並於第 5 節中探討相關實驗。最後在第 6 節提出本篇論文的結論與未來展望。

2、資料表示法

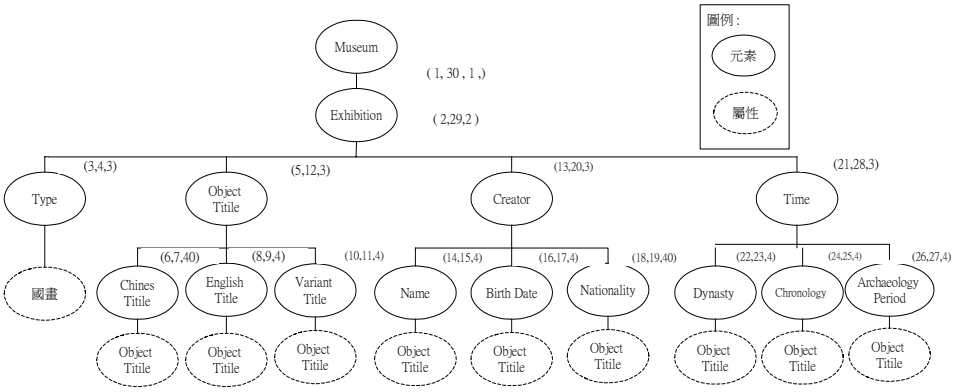
本節將說明本論文針對 DTD、XML 文件以及 XQuery 查詢句的表示法，以及其他相關定義。

2.1 DTD Tree

DTD 為一種定義 XML 資料型態之格式，我們將其轉換成 DTD Tree 以便清楚顯示其巢狀結構。本篇論文將以精簡的史博館版畫藏品之 DTD 為例做說明。如圖一所示，Museum 元素可有多個 Exhibition 元素，在此我們以*表示可出現多個元素，每個 Exhibition 元素主要由 Type、Object Title、Creator、Time 四個元素所組成，分別描述展覽品的種類、名稱、作者以及年代。其中，Object Title 又可分為中文標題、英文標題以及其他標題，作者主要記錄了姓名、生日以及國籍，而年代則記錄了朝代、公元別以及考古別。我們經由 Preorder 的順序走訪 DTD Tree 給予每個元素一個編碼，稱做「路

元素名稱	路徑編碼	元素名稱	路徑編碼	元素名稱	路徑編碼	元素名稱	路徑編碼
Museum	1	Exhibition	2	Type	3	Object Title	4
Chinese Title	5	English Title	6	Variant Title	7	Creator	8
Name	9	Birth Date	10	Nationality	11	Time	12
Dynasty	13	Chronology	14	Archaeology Period	15		

圖二：元素名稱與路徑編碼對應表



圖三：XML 文件之樹狀表示法與元素編碼值

徑編碼」。路徑編碼可用來代表 DTD Tree 中的特定節點，並記錄 Root 走訪到該元素的之路徑。如圖一所示，路徑編碼 2 對應到 Exhibition 節點且所記錄的路徑為 Museum/Exhibition。為了方便日後查詢使用，我們建立元素名稱與路徑編碼對應表，以記錄每個元素所對應之路徑編碼。圖二則為圖一之元素名稱與路徑編碼對應表。

2.2 XML Tree

為了便於對 XML 文件進行查詢處理，在此我們將 XML 文件以樹狀的方式表示，稱做 XML Tree，圖三表示的為符合圖一 DTD 範例之 XML 文件。我們並針對 XML Tree 中的每個元素進行編碼，以便能在查詢處理的過程中有效率的判斷元素之間結構上的關係，並快速的將元素的值抓取出來。編碼的方式為每個元素給予 Start、End 以及 Level 三個編碼值，其中 Start 與 End 分別表示該元素在 XML 文件中 PreOrder 以及 PostOrder 走訪的順序，另外 Level 則記錄該元素在 XML Tree 中的深度。

對每個元素進行編碼之後，將元素在 XML Tree 中的「實體路徑」對應到 DTD Tree 中的路徑，得到該元素之路徑編碼後，再將相同路徑編碼的元素收集起來，以此將元素分類表示。XML 文件中的實體路徑除了可以透過元素名稱表示之外，另外也可藉由元素的 Start 編碼值表示。以圖三為例做說明，Archaeology Period 在圖中的實體路徑為 Museum/Exhibition/Time/Archaeology Period，而透過 Start 編碼值得表示方式為(1, 2, 21, 26)。此實體路徑對應到圖一之 DTD Tree 得到之路徑編碼為 15，因此將 Archaeology Period 元素跟其他路徑編碼同為 15 之元素統一在一個集合表示。

2.3 Query Tree

XQuery 主要由路徑表示法所組成，路徑表示法表示了元素與元素之間結構上的關係。以 books/book 為例，books 與 book 為元素名稱，而 '/' 則代表父子關係，表示我們希望取出 books 元素下一層的所有 book 元素。查詢樹主要是將查詢句中的所有路徑表示法用樹狀的方式加以表示，其中在查詢樹中的每條路徑皆對應到 XQuery 查詢句中的路徑表示法，且每個節點與 Edge 皆符合路徑表示法中節點與節點間結構上的關係。舉例說明，圖四(a) 之查詢句主要是取出 Museum 底下所有 Type 為國畫的 Exhibition 元素，並回傳該展覽品之中文名稱以及作者姓名。圖四(b)則將該查詢句轉換成查詢樹，原先查詢句中 For 子句中的路徑表示法為'/Museum/Exhibition' 轉換成 Museum 以及 Exhibition 兩個節點，並用實線連接表示兩個節點間為父子關係，Where 子句與 Return 子句的路徑表示法將分別將 Title, ChineseTitle 與 Name 加到 Exhibition 節點底

下。在本篇論文中，元素間的限制可以為“/”或“//”。

我們針對查詢樹中的每個節點，分別指定不同的型態，以利後續的查詢處理。節點的型態共計分為 6 種，每個節點規定只能符合其中一種型態，各節點定義的優先順序為下列所示：

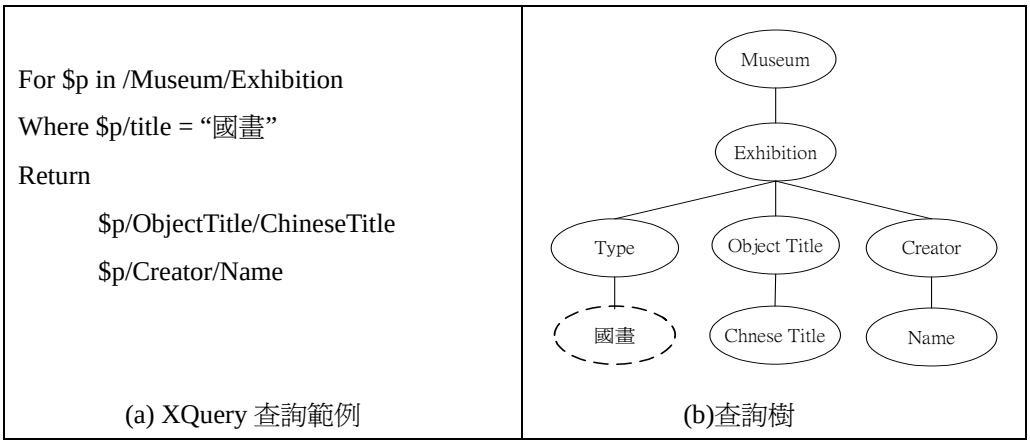
1. 數值節點 (VN) ：為查詢樹中記錄文數值限定的節點，如：圖四(b)中 Type 節點下的 ’國畫’ 節點。
2. 回傳節點 (RN) ：為在 XQuery 中所指定要回傳的節點，如圖四(b)中 ObjectTitle 底下的 ChineseTitle 節點以及 Creator 底下的 Name 節點。
3. 葉子節點 (LN) ：查詢樹中每條路徑最深的節點，但不為 VN 或 RN。如果 ChineseTitle 不為 RN，則其型態為 LN。
4. 黏合節點 (GN) ：為查詢樹中每個有分支的節點。如圖 中的 Exhibition 節點。我們也將根節點指定為此類型。
5. 孫子節點 (DN) ：為查詢樹中與其下一個節點為祖孫關係的節點。
6. 孩子節點 (CN) ：查詢樹中與其下一個節點為父子關係的節點，我們皆將之稱為 CN，如圖中 ObjectTitle 與 Creator 節點。

在之後的討論中，若節點的型態符合 GN、RN 或是 LN，我們統稱該節點為 GRL 節點。

3、分解查詢樹

本節將說明在對查詢句做查詢處理之前的前置處理部份，其主要是將查詢樹切割為若干個「片路路徑」，然後給予每個片段路徑一個「資料流」用來存放XML文件中符合該片段路徑之元素。在此我們採用Bottom Up的方式建立片段路徑。在建立片段路徑的過程中，主要先找出所有第一個節點與最後一個節點皆為GRL節點或DN，且路徑中的所有節點皆「不為」GRL節點或DN所組成之路徑，再由這些路徑當中取出第二個節點到最後一個節點形成片段路徑。因查詢樹中共定義六種節點型態 GN , RN、LN、DN、CN 以及 VN，又片段路徑中的所有節點皆不為 GN、RN、LN 或 DN，故只可能為 CN 或 VN，且又因為 VN 只能出現在路徑的最後一個節點，因此片段路徑中的所有節點皆為 CN。如此一來，所切割出來的路徑，除了開頭的元素，其餘元素之間皆為父子關係。譬如，圖四(b)中的查詢樹，將切割出四個片段路徑 /Museum/Exhibition 、//Type 、 //ObjectTitle/ChineseTitle 與 //Creator/Name 。其中，Root 元素開頭的片段路徑第一個 Edge 將用”/” ，其餘皆用”//” 表示，來標示該路徑不是從 Root 開始。

接下來，針對某個片段路徑，我們首先透過 DTD Tree 找到對應的路徑編碼，再從 XML 文件中找出所有符合該路徑編碼之元素，將其



圖四：XQuery 查詢範例與對應之查詢樹

存放在該片段路徑的資料流中。以//Creator/Name 為例做說明，首先將找出最後一個節點 Name 在圖二中對應之路徑編碼，接著檢查該路徑編碼在 DTD Tree 所表示的節點之父節點是否為 Creator，若是的話該路徑編碼則為符合//Creator/Name 之路徑編碼，在此將為 9。而 XML 文件中符合路徑編碼為 9 的元素為(14 ,15 ,4)，因此將(14 ,15 ,4)存放在//Creator/Name 的資料流中。為了表示方便，在本篇論文中資料流中的資料將僅以元素的 Start 編碼值做代表，而每個片段路徑資料流中的元素我們稱之為「中繼資料」。圖五中顯示了四個片段路徑資料流中所存放的元素編碼值。

片段路徑	資料流中的元素編碼值
/Museum/Exhibition	2
//Type	3
//ObjectTitle/ChineseTitl	6
//Creator/Name	14

圖五：片段路徑存放之元素編碼值

4、組合中繼資料

根據前置處理的結果，本節說明查詢處理的主要過程，其中可細分為建立片段接合路徑與結合片段接合路徑兩個部份。

4.1 建立片段接合路徑

在此，我們希望組合「片段路徑」以形成「片段接合路徑」。片段接合路徑主要由兩個或兩個以上的片段路徑所組成，而除了第一個片段路徑的最後一個節點，與最後一個片段路徑的最後一個節點不可為 DN 之外，其餘片段路徑的最後一個節點必須要為 DN。如圖六所示為建立片段接合路徑之演算法，主要是由查詢樹中的 GRL 節點開始向上找到第一個 GN 或 RN (L01 ～L04)，同時將尋找過程中所走訪的節點依序存入 List 中 (L05~L08) (List 為一個串列結構，用來存放查詢樹中的節點，，而 List 中的 GRL 節點皆有一資料流，如第 3 節所討論)。等走訪到第一個 GN 或 RN 時，我們則將 List 中所紀錄的資訊傳到 Build_Single_Path 演算法以建立片段接合路徑(L09)。Build_Single_Path 主要是藉由每個元素的 Start 與 End 編碼值，來判斷資料流中的元素是否可組合成符合片段接合路徑的實體路徑。也就是，若元素 A 與 B，A.Start < B.Start 且 A.End > B.End ，則 A 與 B 可組合成合理的實體路徑。最後，每個片段接合路徑的最後一個查詢節點，將給予一個 GFP_Table 用來存放片段接合路徑的結果，其中合理的實體路徑以第一個與最後一個片段路徑的最後一個元素表示。

以/Museum/Exhibition 與//Type 為例做說明，其資料流中的資料分別為 2 與 3，且這組資料可接合為/Museum/Exhibition/Type，因此以(Exhibition , Type)記錄符合/Museum/Exhibition/Type 片段接合路徑的結果。圖七顯示將圖五之片段路徑接合之結果。

```
L01  For (each N in QueryTree) {
L02    If (N.type=GN、RN 或 LN) {
L03      List.add(N)
L04          PN=N.parent
L05  While (PN.type!=GN || PN.type!=RN || PN!=NULL)
L06          { If (PN.type=DN) List.add(PN)
L07               PN=PN.parent
L08          }
L09           GFP_Table=Build_Single_Path(List)
L10           N.GFP_Table= GFP_Table
L11           List.clean()
L12     }
L13  }
```

圖六：建立片段接合路徑之演算法

片段路徑	片段接合路徑	合理的實體路徑
/Museum/Exhibition 、//Type	/Museum/Exhibition/Type	(2,3)
/Museum/Exhibition、//ObjectTitle/ChineseTitle	/Museum/Exhibition、/ObjectTitle/ChineseTitle	(2,6)
/Museum/Exhibition、//Creator/Name	/Museum/Exhibition/Creator、/Name	(2,14)

圖七：片段接合路徑與記錄之資訊

4.2 結合片段接合路徑

在此我們將之前產生的片段資訊，組成符合整個查詢樹的資料。圖八列舉了結合片段路徑之演算法。因符合片段接合路徑的實體路徑皆存放於片段接合路徑最後一個節點的 GFP_Table 內，而又因片段接合路徑的最後一個節點為 GRL 節點，因此我們首先將查詢樹中所有的 GRL 節點形成一個集合，稱之為 QnodeSet。基本上組合的順序採 PostOrder 或 Preorder 皆可，在此用 Preorder 的方式進行結合。首先我們將 QnodeSet 集合內的節點，依照在查詢樹中 Preorder 出現的順序做排列，以便我們可以用 Top Down 的方式來結合片段接合路徑，形成最後的查詢樹(L01~L02)。接下來，我們依序取出 QnodeSet 集合中的節點做處理。在處理的過程中，將透過 Result 存放需要進行比對的資料。其中，當取出來的節點為 GN 時則將 Result 設為 NULL，若不為 GN 則將該節點 GFP_Table 中的紀錄存放於 Result 中(L05~L06)。

```
L01  QnodeSet = GetGRLNodeFromQueryTree();
L02  CQN      = QnodeSet.getNodebyPreOrder();
L03  For (each GT in CQN.GFP_Table) {
L04        For (each T in GT) {
L05              If (CQN.isGN())       Result = NULL
L06              Else   Result = T
L07              CQN.CurrentNode = T
L08              CombineGRLNodeUnderCurrentNode()
L09        }
L10  }
```

圖八：結合片段接合路徑之演算法

之後比對該節點 GFP_Table 內的紀錄與該節點底下所有 GRL 節點的 GFP_Table 中的紀錄，是否能組成符合整棵查詢樹的答案(L08)。在處理完 QnodeSet 中所有的節點之後，將 Result 回傳則回最後的答案。

以圖六為例做說明，我們可發現元素 3 , 6 , 14 皆在元素 2 底下，因此可找出(2,3,6,14)一組解，最後再將(2,3,6,14)在 XML 文件中對應的元素值抓取出來，回傳給使用者。

5、實驗

在本節中我們將設計兩個實驗分別探討路徑數量以及查詢句回傳正確資料的比率對整體查詢效率上的影響。我們使用的實驗環境採用 CPU 為 Pentium II 400MHz 的個人電腦以及 512MB 的記憶體，加上 Linux Mandrake release 7.0 的作業系統。此外採用 Borland C++ builder 6.0 來實作此系統。

5.1 路徑數量的影響

在本實驗中將探討當使用者所下之 XQuery 的路徑個數越多時，對此系統效率的

```
L01  <!ELEMENT E1(E2+)>
L02  <!ELEMENT E2(E3,E4,E5……..略,E22)>
L03  <!ELEMENT E3(E23,E24)>
L04  ……..略
L05  <!ELEMENT E22(E57,E58)>
L06  <!ELEMENT E23(#PCDATA)>
L07  <!ELEMENT E24(#PCDATA)>
L08  ……..略
L09  <!ELEMENT E57(#PCDATA)>
L10  <!ELEMENT E58(#PCDATA)>
```

圖九：路徑數量實驗使用之 DTD

影響。為此我們先設計一簡單之 DTD，如圖九所示，此 DTD 主要配合史博館版畫類之 Metadata 的設計，所以最長的路徑長度不超過 4，其 E1 元素下可有多個 E2 元素，E2 元素下則有 E3 到 E22 共 20 個元素，其中每個元素底下又分別有兩個元素。我們透過 IBM XMLGenerator 產生 XML 資料，所產生的資料大小約 1MB。在實驗中我們所使用的 XQuery 則如圖十所示，在實驗中我們將固定限制式，而增加回傳路徑的個數。

```
L01  For $p in document(NumberOfPath.xml)/E1/E2
L02  Where $P/E3/E23="E23"
L03  Return $P/E3/E24
```

圖十：路徑數量所用之 XQuery 查詢句

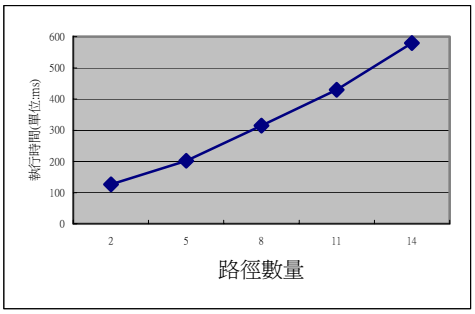
圖十二、圖十三與圖十四分別顯示改變路徑數量後，其分別對執行時間、片段路徑的數量與中繼資料數量的影響。由實驗的結果可得知當路徑數量增加時，查詢過程中所切割出來的片段路徑亦隨之增加，而片段路徑的增加也造成中繼資料的數量隨著成長，因此造成整體執行時間的增加，由此可知中繼資料與片段路徑的數量對整體的執行時間有一定的影響。不過我們也觀察到其曲線約略呈線性成長，所以對系統的效率並不會造成很大的負擔。

```
L01  For $p in document(Museum.xml)/Museum/Exhibition
L02  Where $p/title="國畫"
L03  Return $p/ObjectTitle/ChineseTitle
L04          $p/Creator/Name
```

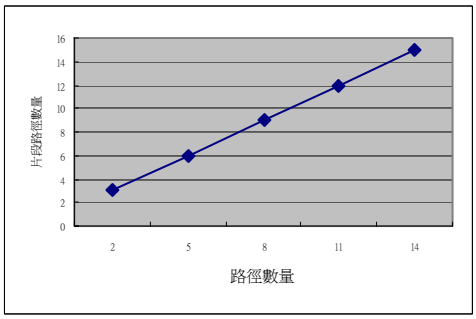
圖十一：回傳資料比率實驗所使用之 XQuery

5.2 回傳資料比率之影響

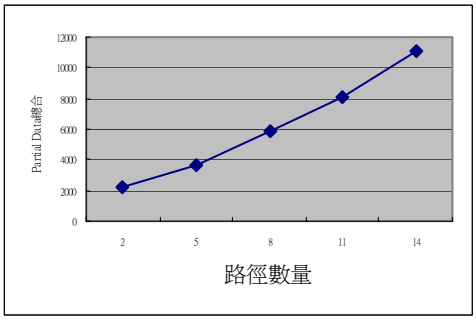
在此實驗中我們將探討，當回傳正確資料的比數增加時對系統之影響。此實驗中我們將採用完整的史博館版畫類之 Metadata 為 DTD，透過 IBM XMLGenerator 產生 XML 資料，版畫的個數固定為 2000 筆，檔案大小為 10MB，且藉由圖十一之 XQuery 查詢句進行查詢。我們增加這些筆數中回傳正確資料的比率，隨著回傳正確資料比率的增加，回傳的資



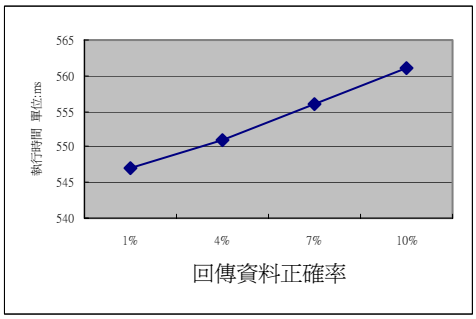
圖十二：路徑數量實驗之執行時間



圖十三：路徑數量實驗之片段路徑數量



圖十四：路徑實驗之中繼資料數量



圖十五：回傳資料比率實驗之執行時間

料將增多，我們所設定的回傳比率分別 1%、4%、7% 以及 10%。由圖十五執行時間之結果可得知，在處理的資料筆數固定的情況下，整體的執行時間將隨著回傳正確資料率的增加而成長，但是其仍然呈現線性的成長。

6、結論與未來方向

由於數位典藏計畫的順利推動，文物的後設資料也不斷的快速成長，如何有效的管理，已是一個重要的課題。本論文中針對最常見的 XQuery，提出了一系列的資料結構與演算法以便從龐大的 XML 資料中快速的取出使用者所需的資料。其主要的概念是將 XQuery 建立成查詢樹，並將其分解成一條條的片段路徑，從 XML 文件中取出符合片段路徑的資料，之後再將這些資料黏合成為符合片段黏合路徑的實體路徑，最後組成整個查尋樹找出符合查詢樹的答案。我們以歷史博物館使用的實際 DTD 進行測試，實驗結果顯示，查詢路徑數量與回傳資料比率，對系統的效率大概呈現線性的影響，成果相當良好。

本論文未來的研究方向，將針對更複雜的 XQuery 進行更深入的探討，譬如巢狀結構的 XQuery 或是有巨集函數的 XQuery，並希望最後整合個別所提出的方法，解決整個 XQuery 的查詢處理問題。

■ 誌謝：此計畫由國科會贊助。
編號 NSC93-2422-H-019-001

參考文獻

[1]中央研究院後設資料工作組
http://www.sinica.edu.tw/~metadata/news/news-frame.html

[2] Shurug Al-Khalifa, H. V. Jagadish, Nick Koudas, Jignesh M. Patel, Divesh Srivastava, Yuqing Wu “Structural Joins: A Primitive for Efficient XML Query Pattern Matching”, In Proceedings of ICDE, Feb. 2002.

[3] Haifeng Jiang, Wei Wang, Hongjun Lu, Jeffrey Xu Yu, “Holistic Twig Joins on Indexed XML Documents”, In Proceedings of the 29th VLDB Conference, Berlin, Germany, 2003.

[4] Shurug Al-Khalifa, H. V. Jagadish, Nick Koudas, Jignesh M. Patel, Divesh Srivastava, Yuqing Wu “Structural Joins: A Primitive for Efficient XML Query Pattern Matching”, In Proceedings of ICDE, Feb. 2002.

[5] Haifeng Jiang, Hongjun Lu, Wei Wang, Beng Chin Ooi , “XR-Tree: Indexing XML Data for Efficient Structural Joins”, In Proceedings of ICDE 2003.

[6] Raghav Kaushik , Pradeep Shenoy, Philip Bohannon, Ehud Gudes, “Exploiting Local Similarity for Indexing Paths in Graph-Structured Data”, ICDE 2002.

[7] Chen Qun, Andrew Lim, and Kian Win Ong, "D(k)-Index: An Adaptive Structural Summary for Graph-Structured Data.", Proc. ACM SIGMOD Int'l Conf. management of Data, Jun. 2003.

[8] Hao He and Jun Yang, "Multiresolution Indexing of XML for Frequent Queries.", Proc. 20th Int'l Conf. Data Eng. (ICDE'04), Mar. 2004.

[9] Yuqing Wu, Jignesh M. Patel, H. V. Jagadish , “Structural Join Order Selection for XML Query Optimization”, In Proceedings of ICDE 2003.

[10] Raghav Kaushik, Rajasekar Krishnamurthy, Jeffrey F Naughton, Raghu Ramakrishnan, “On the Integration of Structure Indexes and Inverted Lists“, SIGMOD 2004.