國立臺灣海洋大學

資訊工程學系

碩士學位論文

指導教授：張雅惠 博士

具有結構及關鍵字限制之 XML 查詢最佳化研究

The Research on Optimizing XML Queries with Structural and Keyword Constraints

研究生：羅誠正

中華民國 100 年 7 月

具有結構及關鍵字限制之 XML 查詢最佳化研究

The Research on Optimizing XML Queries with Structural and Keyword Constraints

研 究 生：羅誠正　　Student：Cheng-Cheng Lo

指導教授：張雅惠　　Advisor：Ya-Hui Chang

國 立 臺 灣 海 洋 大 學

資 訊 工 程 學 系

碩 士 論 文

A Thesis

Submitted to Department of Computer Science and Engineering

College of Electrical Engineering and Computer Science

National Taiwan Ocean University

In Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in

Computer Science and Engineering

July 2011

Keelung, Taiwan, Republic of China

中華民國 100 年 7 月

國 立 臺 灣 海 洋 大 學

博碩士論文紙本及全文上網授權書

（提供授權人裝訂於紙本論文書名頁之次頁用）

本授權書所授權之論文為授權人在國立臺灣海洋大學
資訊工程學系_____組 99 學年度第____學期取得碩士學位之論文。

記錄編號：G0M97570028
論文題目：具有結構及關鍵字限制之 XML 查詢最佳化研究
指導教授：張雅惠

茲同意將授權人擁有著作權之上列論文全文電子檔(含摘要)，依下述授權範圍，以非專屬、無償授權國立臺灣海洋大學圖書館，不限地域、時間與次數，以微縮、光碟或其他各種數位化方式將上列論文重製，並得將數位化之上列論文及論文電子檔以上載網路方式，提供讀者基於個人非營利性質之線上檢索、閱覽、下載或列印。

■ 讀者基於非營利性質之線上檢索、閱覽、下載或列印上列論文，應依著作權法相關規定辦理。

論文全文上載網路公開之範圍及時間：	
校內區域網路	■ 立即公開
校外網際網路	■ 立即公開
紙本公開陳列範圍及時間：	■ 茲同意將本著作立即公開陳列上架，並同意因遺失或損毀重製。

授 權 人：羅誠正

學　　號：M97570028

E-mail ：law326@gmail.com

親筆簽名或蓋章：_____

中　　華　　民　　國　　年　　月　　日

國家圖書館
博碩士論文電子檔案上網授權書

本授權書所授權之論文為授權人在國立臺灣海洋大學資訊工程學系 99 學年度第___學期取得碩士學位之論文。

記錄編號：G0M97570028
論文題目：具有結構及關鍵字限制之 XML 查詢最佳化研究
指導教授：張雅惠

茲同意將授權人擁有著作權之上列論文全文（含摘要），非專屬、無償授權國家圖書館，不限地域、時間與次數，以微縮、光碟或其他各種數位化方式將上列論文重製，並得將數位化之上列論文及論文電子檔以上載網路方式，提供讀者基於個人非營利性質之線上檢索、閱覽、下載或列印。

□ 上列論文為授權人向經濟部智慧財產局申請專利之附件或相關文件之一（專利申請案號：　　），請於　年　月　日後再將上列論文公開或上載網路。

□ 因上列論文尚未正式對外發表，請於　年　月　日後再將上列論文公開或上載網路。

授權人：羅誠正

親筆簽名及蓋章：_____　　民國　年　月　日

身分證字號：
電話：　　傳真：
聯絡地址：
E-Mail：law326@gmail.com

摘要

如何有效率地查詢 XML 文件，早已成為現今重要的研究議題。在本論文中，我們針對具有結構及複雜資訊檢索限制的 XQFT 查詢句，加入查詢處理最佳化的技術，以找出最有效率的查詢組合。

在論文[吳 09]裏針對該類查詢句提出兩種查詢處理方式，分別稱為「結構優先」和「關鍵字優先」 。結構優先的方式是先依建立的查詢樹，抓取出符合標籤限制的元素資料，接著判斷是否符合關鍵字限制，再將此部份結果合併成符合整體樹狀結構限制的結果。關鍵字優先則是依據查詢樹中的關鍵字，先取出符合關鍵字的元素資料，接著計算這些元素的最低共同祖先(Lowest Common Ancestors)，並且判斷是否符合關鍵字限制以及路徑限制，最後再將結果合併。為了執行最佳化處理，我們將原有的系統切割成數個元件，並設計了對應的成本模型以及改寫法則，利用各個元件不同的先後排列組合，來找到最快速的查詢方法。

我們透過一系列的實驗，來證明提出的成本模型正確性，以及本系統的確能有效率地找出最快的執行策略。

Abstract

Efficient query processing on XML data has become a very important research issue. In this paper, we consider the XML query with structural and keyword constraints and apply the query optimizing mechanism to improve the efficiency of query processing.

Two approaches are proposed in the thesis [吳 09], that is, the structure-first approach and the keyword-first approach. The structure-first approach will first identify the elements that satisfy the tag constraint from the query tree, and then process the full-text constraint. The satisfied elements will be combined to meet the complete twig constraints. On the other hand, the keyword-first approach will first identify the elements which represent the required keywords, calculate the LCA for those elements, and then return the elements which satisfy the given full-text predicates and structural constraints. We design several components based on the two approaches. We also propose the corresponding cost model and rewriting rules, to choose the most efficient plan.

To show the effectiveness of the proposed cost model and the efficiency of our system, we have performed a series of experiments. The results show that our system can indeed produce the best execution strategy.

目錄

圖目錄

表目錄

第一章 序論

1.1 研究背景與目的

由於近年來網路的蓬勃發展以及普及，使得資訊間的傳遞更加的便利，各種不同的資料格式也因此產生，但不同的資料格式卻造成了資料流通上的麻煩。因此 W3C 所制定的 XML(Extensible Markup Language)因為可以允許使用者自定文件所需要的標籤和結構，並且利用 DTD(Document Type Definition)或 XML Schema 來規範其定義，再加上 XML 是使用純文字的方式儲存，檔案大小較小，因此逐漸成為各種資料常用的表示方法。

然而也正因為 XML 的普遍流行，所以如何有效率的查詢 XML 變成為一個重要的議題。針對 XML 文件的查詢，W3C 先後提出了 XPath 和 XQuery 來方便使用者查詢，近幾年並進一步提出 XQFT[XQFT]語法來處理兼具結構限制和資訊檢索限制的查詢句。針對該語法，論文[吳 09]提出了兩種做法，一種是「結構優先」(Structure-first)，另一種則是「關鍵字優先」(Keyword-first)。在結構優先的作法上，是先針對輸入的查詢句(XQuery)建立查詢樹，接著根據該查詢樹抓取 XML 文件中符合查詢樹節點的元素，然後處理資訊檢索限制，並將最後的結果利用 TJFast[LLCC05]合併。至於「關鍵字優先」則是根據查詢樹中的資訊檢索限制，取得符合具有該關鍵字的元素，判斷資訊檢索限制後，再判斷路徑限制，並且同樣利用 TJFast 將結果合併。不過該論文並沒有辦法自動找出最佳的執行策略。

因此，本論文研究的目的，便是希望找出兩者在不同查詢狀況下各自的優點，來提升整體查詢的效率。我們會將這兩種作法切割成幾個不同的運算子(Operator)，試著利用將不同運算子的執行順序重新排列，來找出最佳的執行計畫，使系統可以有最佳的執行效率。

1.2 研究方法與貢獻

由於查詢系統裏包含「結構優先」和「關鍵字優先」兩種做法，且各自包含數個運算子，因此我們利用建立 cost model 的方式，使得系統可以正確的找出最佳的方法以及排列組合。我們利用對每個運算子演算法的時間複雜度分析，來建立 cost model，並以實驗的方式取得對應的係數，讓每個運算子的 cost model 可以忠實的呈現執行的效率。

至於 cost model 中，所需要的輸入輸出資料量，我們在 XML 文件的前處理時，便搜集了對應的統計資料。另外，我們也設計了數個 rewriting rule 產生許多不同順序的執行策略，再利用我們設計的 cost model，從中選出 cost 最低的 plan 來執行。

最後，本論文的貢獻如下：

1. 將「結構優先」與「關鍵字優先」各階段的演算法模組化，定義成數個 Operator，讓這兩個作法不同執行階段可以更加彈性，並且能夠調整。
2. 我們定義了 rewriting rule 讓系統可以針對兩種不同的作法，產生一系列不同的執行計畫。
3. 我們同時也對每個 Operator 定義了 cost model 讓系統能夠透過這些 cost model 有效率的找出最佳的執行計畫。
4. 我們實作了這套系統，並且透過一系列的實驗，證明系統的正確性以及效率。

1.3 相關研究

在針對「結構優先」查詢處理的研究方面，[吳 09]利用了[LLCC05]中的 twig join 演算法。[LLCC05] 設計的編碼格式為 Extended Dewey，經由該論文所提出的 Finite state transducer，可以從單一元素的編碼，推導出從 root 到此元素路徑中的所有元素名稱。該論文提出的 twig join 演算法 TJFast，因為只需要存取葉節點，所以很有效率。

在針對「關鍵字優先」查詢處理的研究方面，[吳 09]則是利用[ACD06]提出的作法。[ACD06]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。

在針對 XML 查詢最佳化處理的研究方面，在[MW99]中，討論如何將資料庫系統 Lore 改進成為能夠完全支援 XML 的資料庫系統，並且將原本只適用於關聯式資料庫的統計資料改進成為可以完全支援 XML，同時利用統計資料來提供之後最佳化處理所需要的分數資訊。研究[WPJ03]考慮查詢句中有多個跳層的狀況，如 a//b//c。作者提出五種不同的運算順序：DP、DPP、DPAP-EB、DPAP-LD、FP，其中 DPP 能花較少的時間找到最好的 structural join 的順序，而 FP 則是能花更少的時間找到一個較好的(但並不一定是最好的)structural join 順序。研究[ZC03]則提出一個 GTP(generalized tree pattern)用來表示 XQuery，主要是將查詢句中的結構限定以一樹狀結構體表示之，另外再搭配其他的限定式來限定其樹中每個節點限制，如：標籤名稱，文數值限定等等。利用 GTP，系統便容易產生 physical plan，並在有 schema 的情況下作最佳化。而在[BCFH05]中，作者利用關聯式資料庫中查詢最佳化的 Vertical Partitioning 技術對 XML 的資料進行切割。作者先將 XML 文件用 Tree 的方式表示，將每條路徑所對應的 Data value 儲存起來，並藉由 skeleton 表示 XML 文件的結構。接著將 XQuery 以 graph 表示，並提出一個 Graph reduction 的演算法，針對該 Graph 進行化簡，也就是一次處理一個 Edge，且在處理的過程中，紀錄符合限制的相關資訊，當化簡完所有的 Edge 時，則將答案輸出。

論文[GORS08]則是擴充 W3C 查詢語言的定義，以允許 side effect 的產生。該論文並探討此新的 algebra 對 optimization 的影響，以及在哪些情況下 rewriting rules 仍然適用。[GCV09]便針對 XPath 的查詢方式，建立了一套以 cost model 為基礎的最佳化系統，並且提出了對應 XPath 的 XAlgebra 來處理最佳化的問題，並且利用一系列的 rewriting rule 來產生等價的計畫。另外，該論文也針對 XML 文件建立了可以取得統計資料的 API，在計算 cost 時，即可利

用對於某一路徑的出現數量，或是路徑當中所下的邏輯判斷式的 Selectivity 來計算分數。在[KBMK09]中，則是討論如何在 Run-time 進行查詢最佳化，其方法是先對 Query 建立 Join Graph，並從圖中分數最低的 Operator 開始執行，然後利用 Sampling 的方法，預先取得下一個 Operator 的資料量，然後再進行評估，選擇出可走的路徑中最佳者，其中，為了提升效率，在做 Sampling 的時候，會計算出適當的擷取數量，以避免取得過多的資料造成效率降低。[MN10]則是針對 XPath 設計了一個自動狀態機，並且利用對 XML 文件樹所建立的索引資料進行走訪，利用不同的走訪方式，找出最佳的 XPath 執行順序。

1.4 論文架構

本論文其餘各章節的架構如下:在第二章我們將簡介在此論文中所需要用到的相關定義，如：XML 文件、XQuery、對 XML 文件編碼、XQuery 文法範圍，並對本論文所要解決的問題進行定義，同時也會簡單說明論文當中會使用到的兩個作法。第三章則是說明系統所用到的資料表示方法，包括針對兩個方法所建立的索引資料、統計資料，以及這些資料與查詢樹的關係。第四章我們將會先介紹我們的系統架構，接著在說明所設計的 Physical Operator 以及其對應的 cost model，同時也會說明 cost model 中的係數是如何被找出來。接著會說明我們所設計的 rewriting rule，以及如何產生所有等價執行計畫的演算法，並且會利用幾個範例來演示。而第五章我們將用一系列的實驗來分析系統的效率。最後在第六章我們將提出結論以及未來的研究方向。

第二章 系統架構

在本章中，我們將會說明整個系統的架構、基本的資料表示，另外還會說明如何建立系統所需要的統計資料，以及介紹所有 Physical Operator 的功能，同時也會針對我們要處理的問題，做一些基本的問題描述和定義。

2.1 XML 資料表示

由於我們是以 XML 文件為我們所要查詢的資料，因此，我們首先先介紹 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 2.1 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。

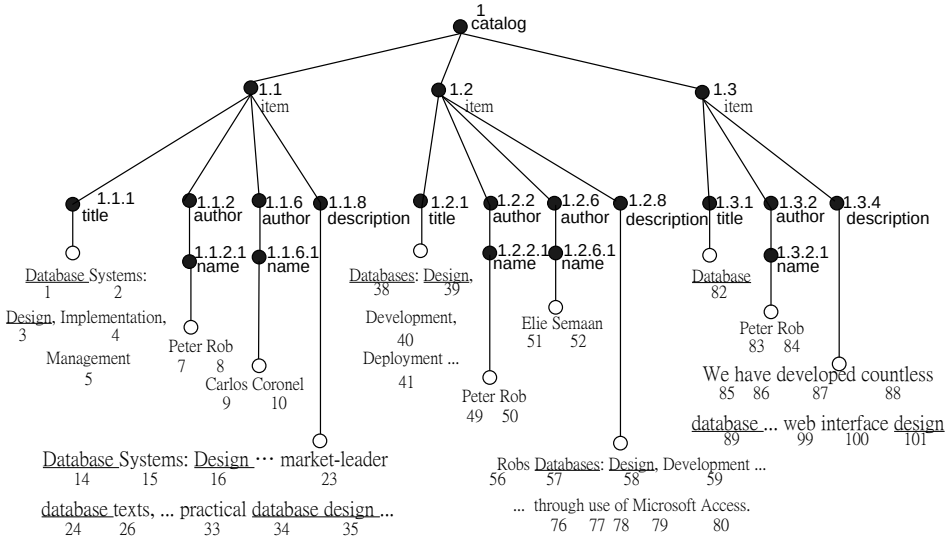

圖 2.1：XML 樹

為了可以快速的推斷元素間的結構關係，我們對 XML 內每個元素進行編碼。我們採用「延伸杜威編碼」(Extended Dewey Code)，其好處是，可以直接將編碼轉成樹根到樹葉標籤路徑(Labeled Path)。

延伸杜威編碼的編碼公式，必須利用對應 XML 文件的 DTD 資訊， 以下公式會計算某個 tagname 其編碼的最後部份：

$$x = \begin{cases} \left\lfloor \frac{y}{n} \right\rfloor * n + k, if\ (y \bmod n) < k \\ \left\lfloor \frac{y}{n} \right\rfloor * n + k, otherwise \end{cases}$$

公式(1)

x: 該元素編碼最後的部份

y: 該元素在 XML 樹中哥哥節點 (left sibling)編碼最後的部份

n: 該 tagname 在 DTD 中共有幾個兄弟

k: 該 tagname 在 DTD 中的兄弟排序 (sibling position)

我們以圖 2.1 的第二個 author 元素為例，對應到該 XML 文件的 DTD 如圖 2.2 所示。首先，author 為 item 的第二個小孩，所以 k=2。哥哥節點為 author，編碼為 “1.1.2” ，所以 y=2。該標籤在 DTD 中共有 4 個兄弟，所以 n=4。由於 y mod 4 = 2 和 k 值相同，所以我們採用第二個公式，得到 x=6。然後擴充其父親 item (“1.1”)的編碼，得到 “1.1.6” 。

```
<!ELEMENT catalog (item*)>
<!ELEMENT item (title, author*, publisher, description)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (name)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT publish(#PCDATA)>
<!ELEMENT description(#PCDATA)>
```

圖 2.2：對應到圖 2.1XML 文件的 DTD

2.2 XQuery 相關定義

為了方便從 XML 文件中取得所需要的資料，W3C 已經針對 XML 文件查詢定義了 XQuery。其中根據定義，一個 XQuery 敘述式主要具有三個部分：for 子句、where 子句、return 子句。其中 for 子句讓變數可以遞迴(iterate over)取得一個運算式的結果，而 where 子句則是允對變數做條件的限制，return 子句則是建構新的 XML 元素作為查詢的輸出。另外各子句則是以路徑表示法 (path expression) 表示元素間的結構限制。

假設要對圖 2.1 的 XML 文件查詢出作者 “Peter” 所撰寫的書籍名稱，則可輸入圖 2.3 的 XQuery 範例。

```
L01  For $p in document (“http://dblab.cs.ntou.edu.tw/book.xml”)
L02          /catalog/item
L03  Where $p//name= ‘Peter’
L04  Return $p/title
```

圖 2.3：XQuery 範例

在範例中，L01 到 L02 即為 For 子句，指定所要查詢的 XML 文件位置為 http://dblab.cs.ntou.edu.tw/book.xml，並且指定要處理的元素為文件中位於路徑 “/catalog/item” 的每個 item 元素並指定給變數 p。L03 則是 Where 子句，針對變數 p 所指定的元素進行進一步的限制。在此範例中限制 p 的子孫元素 (Descendants) “name” 內容必須為 “Peter”。最後在 L04 的 Return 子句，則是指定要回傳的元素為變數 p 之下的 title 元素。

XQFT 進一步擴充 XQuery，除了路徑限制外，同樣也具備資訊檢索的查詢方法。在 W3C 的定義中，可以利用 contains text 針對 Query 下資訊檢索限制 Ordered 和 Distance。Ordered 限制式是用來限制兩個關鍵字出現的前後順序，當下此限制式時，第一個的關鍵字必須出現在第二個關鍵字前面，例如圖 2.4 的 L03 中的 ordered 限制即代表關鍵字 database 必須出現在關鍵字 design 之前。另外除了 ordered 限制外，還可以利用 distance 限制式來限制關鍵字之間的距離，

例如圖 2.4 的 L04 中的 distance 限制即代表關鍵字 database 必須和關鍵字 design 之間至少有兩個字的距離。

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

圖 2.4：具有資訊檢索限制的 XQuery 範例

我們將查詢句利用樹狀的方式來表示，稱作查詢樹。該樹能夠表示結構與關鍵字的限制， 如圖 2.5 的查詢樹範例，即為對應到圖 2.4 的 XQuery 查詢句。在該圖中，實線圓圈代表元素節點，粗實線圓圈則代表 return 子句所要回傳的節點。而元素之間的聯接線則可以是 “|” 或 “||” ，分別代表元素間必須具有父子或祖孫的關係。譬如，元素 item 和 name 之間限定為祖孫關係。特別注意到，item 為一個具有分支的節點，因為 item 下有兩條路徑，一條為//name，另一條為/description，此結構限制稱做 “twig pattern” 的限制，需要特別處理。另外，我們也稱每條路徑最深之節點（不包含資訊檢索部分）為「葉節點」(leaf node) ，如 “name”。

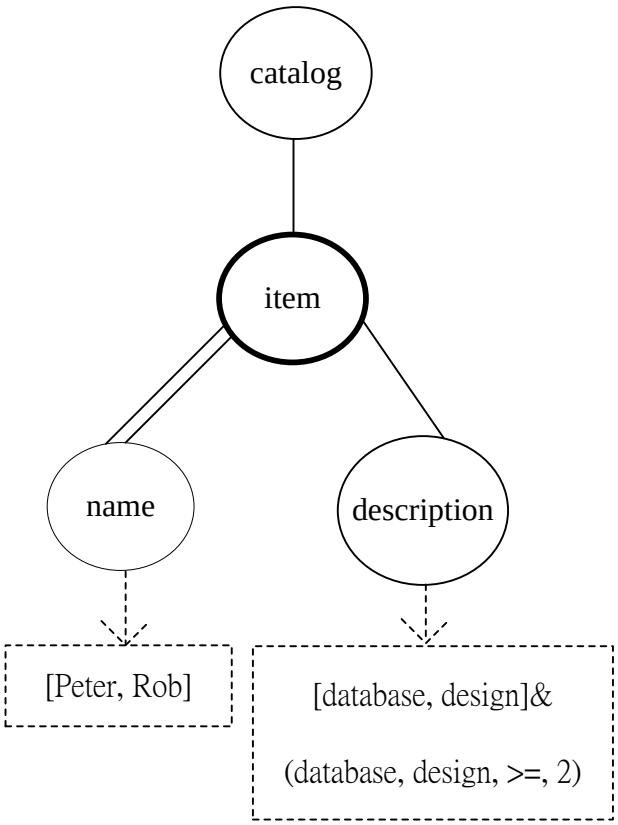

圖 2.5：查詢樹範例

而資訊檢索限制部分，則是表示於葉節點下方虛線方框內 。每一筆 distance 限制式以(term1, term2, operator, number)表示，其中，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。

我們利用 Backus-Naur Form 文法列舉本論文中討論到的 XQuery 範圍，如圖 2.6 所示。

FLOWR::=ForClause+WhereClause?+ReturnClause
ForClause::= “For” (Var “in document(” Expr “)”)
WhereClause::= “Where” (Wi LogicalOp?)+
ReturnClause::= “Return” (Expr)
Expr::=pathExpr
IRWi::= “(” String “ftand” String IROp “)” | IRWi “ftand” IRWi

```
Wi::= Expr “contains text” IRWi
Var::= “$”Literal
Literal::=String | Digits
pathExpr::=pathExpr Axis  String | Axis String
Axis::= “/” | “//”
IROp::= “ordered” | “distance” CompOp Digits “words”
CompOp::= “at least” | “at most”
LogicalOp::= “and”
String::= ‘ “ ’ [A-Za-z] ([A-Za-z0-9] | ‘-’ )* ‘ ” ’
Digits::= [0-9]+
```

圖 2.6：本論文討論的 XQuery 文法範圍

在圖 2.6 中可以看到，我們所處理的 XQuery 除了基本的結構查詢外，同時也在 Where 限制中，加入資訊檢索的條件限制。針對資訊檢索的限制，可以處理 ordered 和 distance 兩個條件。另外在本文中，不討論有 Let 子句以及 return 子句中還有子查詢句的狀況。同時在 Where 子句之間的邏輯判斷，我們不討論 Or 的狀況，只討論 And 的狀況。

2.3 基本定義

在本論文中，為了解決我們所要處理的問題，因此我們提出以下定義：

[定義 1]：T_match(t)：若 XML 樹中某一節點的標籤符合查詢樹中某標籤 t 的限制，則稱該節點為一個 T_match(t)。例如，圖 2.1 中的節點 1.1、1.2 和 1.3，分別符合圖 2.5 查詢樹的標籤限制 “item”，也就是 T_match(item)。

[定義 2]：P_match(p)：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 p，則稱該節點為一個 P_match(p)。例如，圖 2.1 中的節點 1.1.2.1，其標籤路徑為 “/catalog/item/author/name”，為一個 P_match(“/catalog/item//name”)。

[定義 3]：K_match(k)：若 XML 樹中某一節點的內容存在查詢樹中要求的關鍵字 k，則稱該節點為一個 K_match(k)。例如，圖 2.1 中的節點 1.1.1、1.1.4、1.2.1、1.2.4、1.3.1 和 1.3.3，皆為 K_match(database) 。

[定義 4]：C_match(op)：若 XML 樹中某節點的內容符合查詢樹中某個 contains text 資訊檢索限制 op，則稱該節點為一個 C_match(op)。例如，圖 2.1 中的節點 1.1.1、1.1.3、1.1.4、1.2.1、1.2.3、1.3.3 為 C_match(ordered)。

[定義 5]：FT_match(L)：若 XML 樹中某節點的內容符合查詢樹中某個葉節點 L 當中所有的資訊檢索限制，則稱該節點為一個 FT_match(L)。例如，圖 2.1 中的節點 1.1.4 和 1.3.3 皆為 FT_match(description)。1.2.1 和 1.2.4 之所以不符合，是因為 database 和 design 的距離只有 1。

[定義 6]：match tree：若在 XML 樹中的一群節點，每一個點皆為 P_match 或 FT_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 2.1 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 2.5 查詢樹之 match tree。

[定義 7]：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。

2.4 問題定義

目前在文獻上看到，可以處理 XML 資料查詢方法的，可概分為兩類。第一類，我們稱作「結構優先」。其方法主要是要找出符合查詢樹的 match tree。我們擴充其作法處理資訊檢索限制，其中一種作法流程如圖 2.7 所示，也就是先擷取出 XML 文件中符合查詢句內出現的標籤元素，接著再針對這些元素進行資訊檢索限制判斷，然後再判斷是否符合路徑限制。最後將結果進行 twig 結構的判斷再整合起來。

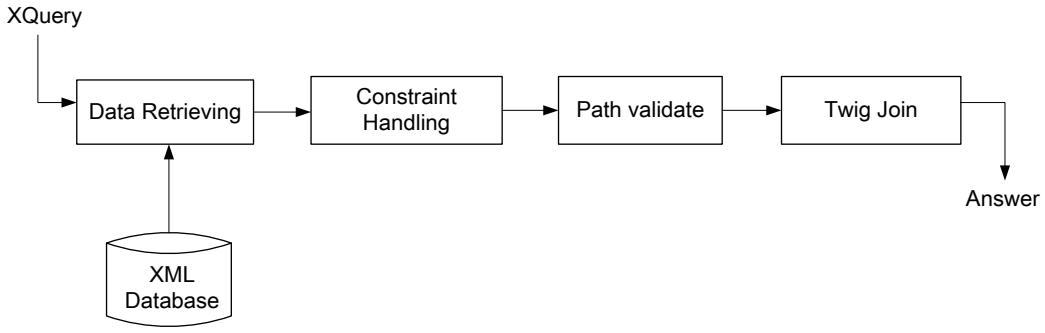

圖 2.7：結構優先的處理範例

第二類作法，我們稱作「關鍵字優先」。該作法支援只以關鍵字下達查詢句。我們擴充其作法來支援同時包含結構限制和資訊檢索限制的查詢句。其作法之一如圖 2.8 所示，也就是先在 XML 文件中找尋符合查詢句內的資訊檢索限制之元素，接著找出這些元素的 LCA(Lowest Common Ancestors)之後，再針對這些元素進行資訊檢索限制的判斷，然後判斷是否符合路徑，最後一樣利用 Twig Join 將結果整合起來。

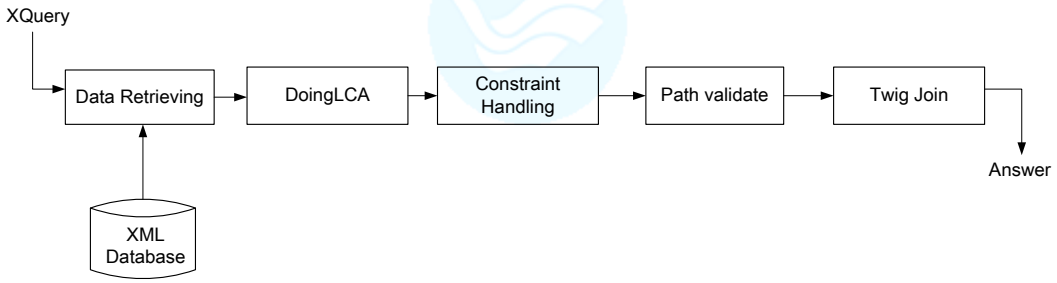

圖 2.8：關鍵字優先的處理範例

而本論文主要的目的，就是要建立一個系統，能夠支援結構優先和關鍵字優先兩種作法，同時利用查詢最佳化的技術，來決定選擇何種執行策略。

第三章 資料表示

3.1 索引與統計資料

在我們的系統中，同時支援兩種不同作法：結構優先、關鍵字優先。為了加快資料擷取的速度，針對結構優先以及關鍵字優先的處理，我們分別用了兩種不同的方式來建立索引值(Index)。在本節中，將會介紹這兩個資料結構在系統中的表示方式。另外，我們也會討論為了支援最佳化處理所搜集的統計資料。

Extended DeweyID	Tagname	Term	Position
1.1.1	title	Database	1
1.1.1	title	System	2
1.1.1	title	Design	3
1.1.1	title	Implementation	4
1.1.1	title	Management	5
…	…	…	…
1.1.4	description	Database	14
1.1.4	description	System	15
1.1.4	description	Design	16
1.2.1	title	Database	38
1.2.1	title	Design	39
1.2.1	title	Development	40
1.2.1	title	Deployment	42
1.2.4	description	Robs	56
1.2.4	description	Databases	57
1.2.4	description	Design	58
1.2.4	description	Development	59
…	…	…	…

圖 3.1：XML 各元素及對應關鍵字表格

首先我們會先將 XML 文件中的每個元素以及各節點內的關鍵字建立成一個表格，如圖 3.1 所示，表格的內容包括：Extended Dewey ID、元素標籤(Tagname)、關鍵字(Term)、關鍵字位置(Position)。在處理文字內容時，我們按

照一般 IR 處理的習慣，針對文字內容做 stopwords 的處理，將 stopwords 替除掉，以免資料過大。

為了支援結構優先，我們會利用圖 3.1 的「元素標籤」欄位來作為主鍵(Key)建立索引資料，並且將索引資料利用 B-Tree 建立起來，如圖 3.2。在 B-Tree 中的節點，會指到圖 3.1 表中對應的資料。所指到的資料包含下列資訊：延伸杜威編碼(Extended Dewey Code)、 關鍵字(Keyword)以及關鍵字的位置(Position)。

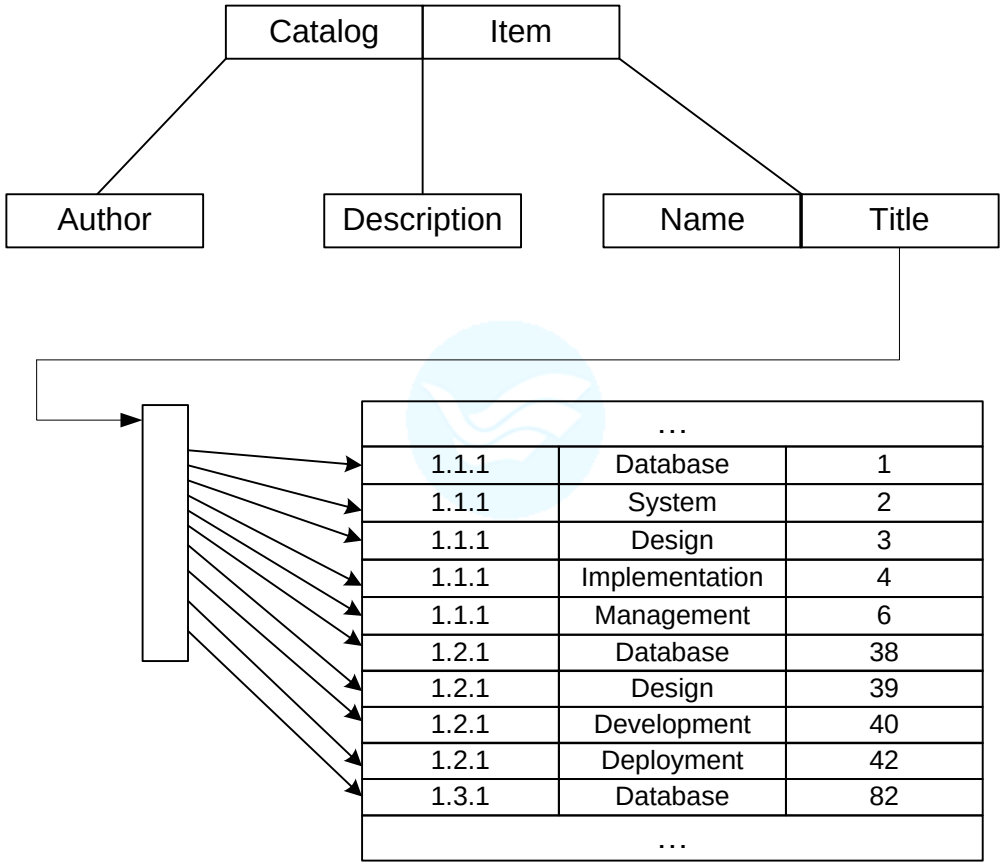

圖 3.2：以圖 3.1 的標籤所建立的索引資料

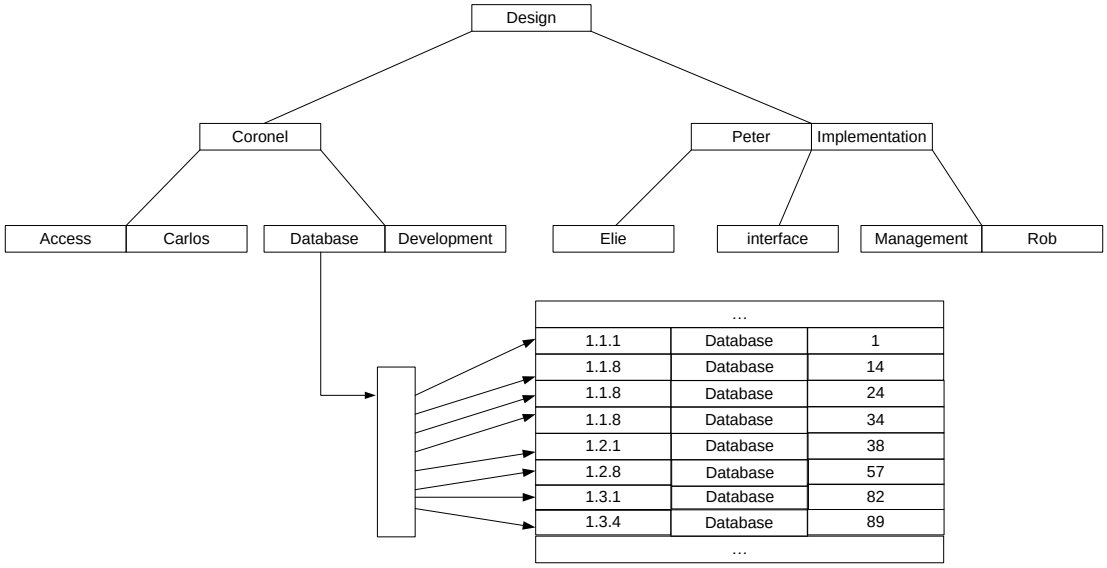

圖 3.3：以圖 3.1 的關鍵字所建立的索引資料

為了支援關鍵字優先，我們也另外建立索引值。我們一樣利用 B-Tree 以圖 3.1 中的「關鍵字」作為主鍵(Key)來建立索引，而回傳的內容則包含：延伸杜威編碼(Extended Dewey Code)、關鍵字位置(Position)。如圖 3.3 所示。

為了在計算 Cost Model 時，可以取得資料實際的統計資訊，以增加計算 Cost 的準確度，在對 XML 文件進行前置處理(Preprocess)的時候，我們同時會統計下列三項資訊：(1) 同一元素名稱(Tagname)出現的次數、(2) 同一關鍵字出現的數量、(3) 對應同一標籤所有元素內關鍵字的個數。例如統計圖 2.1 文件中元素名稱 “item” 的資料個數，我們會先找出符合標籤限制為 “item” 的節點為 1.1 、1.2 和 1.3，因此在此範例中元素 “item” 的統計資料即為 3。另一方面，例如統計圖 2.1 文件關鍵字 “database” 的元素數量，我們會先找到符合限制的節點為 1.1.1、1.1.4、1.2.1、1.2.4、1.3.1 和 1.3.3，因此在此範例中關鍵字 “database” 的統計資料為 6。至於在統計元素名稱為“title”內的關鍵字數量時，因為圖 2.1 中的三個“title”內，分別是 1.1.1 有 5 個關鍵字、1.2.1 有 10 個關鍵字、1.3.1 有 1 個關鍵字，因此“title”元素的關鍵字個數共為 16。

3.2 資料表示

在處理結構優先的狀況下，我們會先要取得 T_match 的資料。取得的 T_match 資料會合併成為一個 Stream，負責記錄符合該節點標籤的元素編碼，接著，我們會將該元素所表示的所有關鍵字串連起來，形成一個屬於該元素的 Term List。因此，一個 Stream 內可能會包含多組 Term List，而每個 Term List 內會記錄關鍵字以及該關鍵字的位置。圖 3.4 為圖 2.4 中的查詢樹對圖 2.1 中的 description 做 T_match 所得到的資料範例。

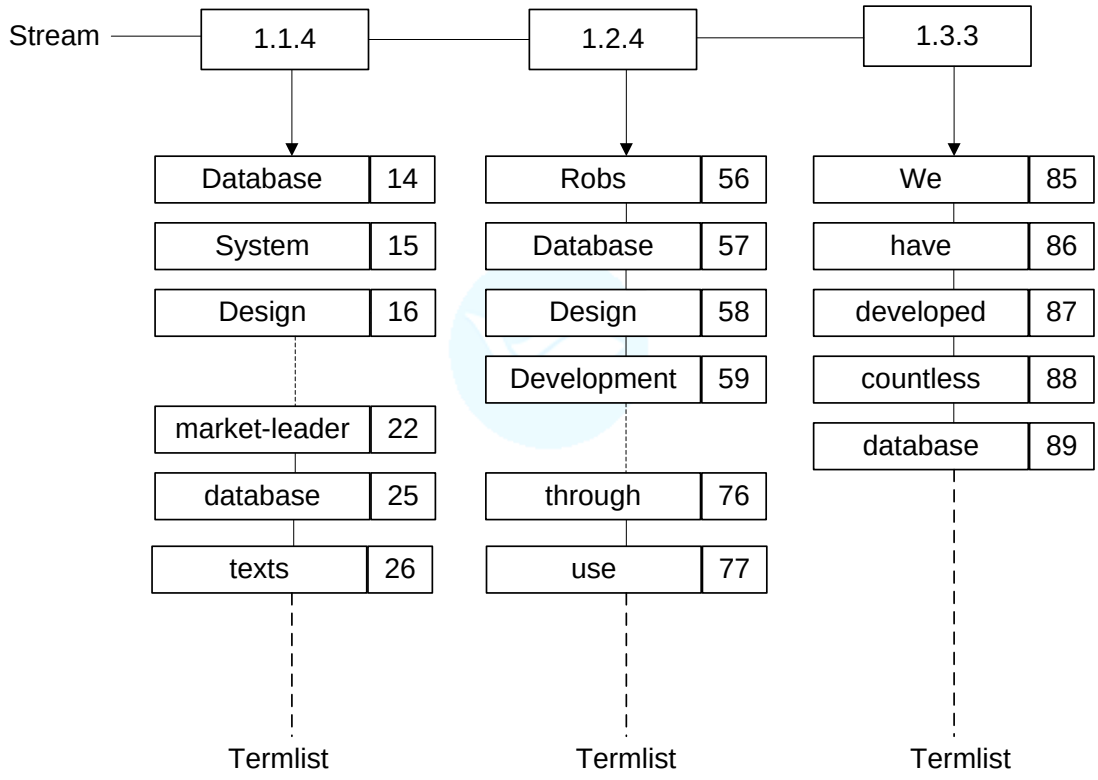

圖 3.4：description 葉節點的 Stream 範例

而當在處理關鍵字優先的狀況下，我們先要取得 K_match 的資料。取得的 K_match 會存成一張 SCU Table (Smallest Containing Unit Table)。SCU Table 中會記錄符合 K_match 的節點杜威編碼，以及此關鍵字的位置。圖 3.5 即為對應關鍵字”database”的索引資料。

Node	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

圖 3.5：database 的關鍵字 SCU 資料

在整體資料結構的表示上，我們會將前面取得的 Stream 或是 SCU 掛回 QueryTree 當中，在之後的資訊檢索限制判斷或是路徑判斷上，我們就可以直接針對各個節點下面的 Stream 或是 SCU 進行處理。圖 3.6 為對應圖 2.4 的 Query 所產生的 QueryTree 以及 Stream 關係圖，在 QueryTree 當中的葉節點對應到的就是在 XML 文件中，符合該葉節點標籤限制的元素，而在各個元素下串起來的，則是該元素內的 Termlist。而圖 3.7 則是對應圖 2.4 的 Query 所產生的 QueryTree 與 SCU 關係圖。與圖 3.8 不同的是，QueryTree 的葉節點所對應到的，是符合該節點資訊檢索限制的 SCU 表。

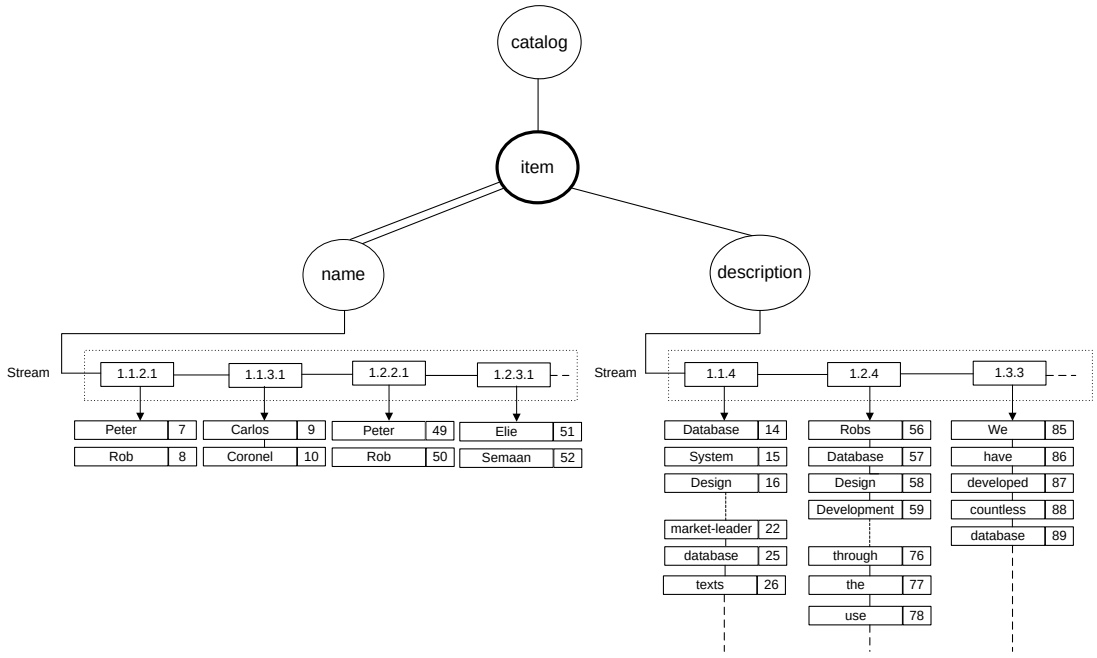

圖 3.6：Stream 與 QueryTree 的關係

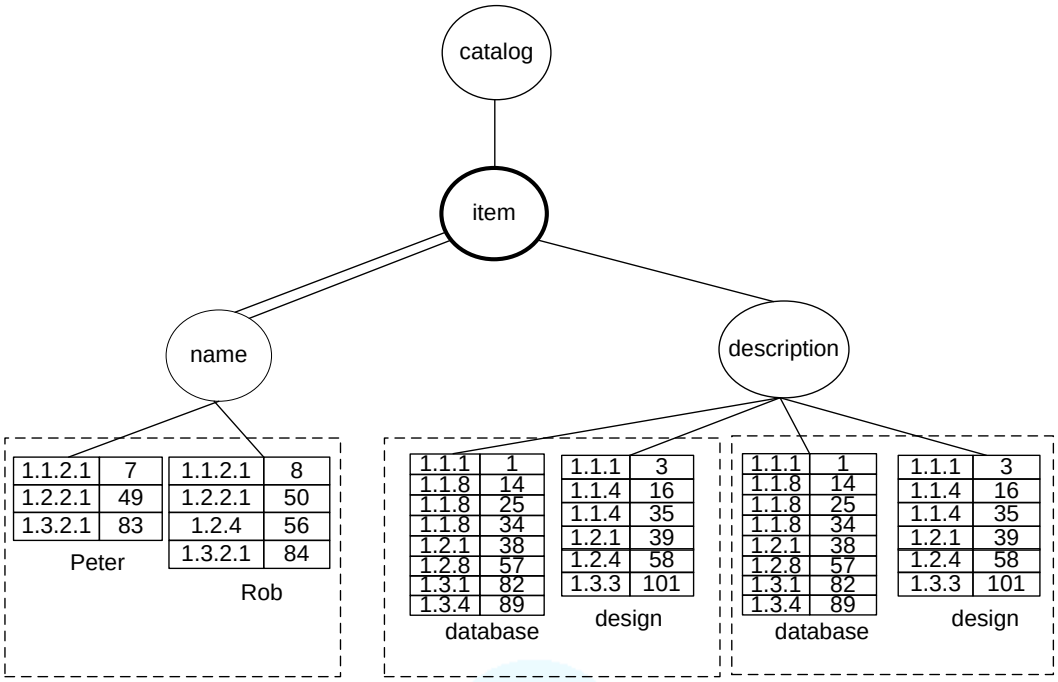

圖 3.7：SCU 與 QueryTree 的關係

第四章 Cost-based Optimization

在本章中，我們將會介紹我們所設計的 Physical Operator 以及其對應的 Cost Model。同時，我們會說明 Cost Model 中所出現的係數是如何被找出來。接著會介紹各個 Rewriting Rule 以及說明如何針對 Query 找出可套用的 Rewriting Rule 演算法。

4.1 系統架構

整體的系統架構如圖 4.1，首先我們會將輸入的 XQuery 查詢句透過 Parser 轉換為預設的執行計畫。接著，再將預設值行計畫送到 Plan Generator 內，套用 Rewriting Rules 並且產生所有可能的組合。然後再將所產生出來的執行計畫集合送到 Physical Plan Selector 內，透過取得統計資料以及利用 Cost Model 計算出來的分數，來選擇最佳的執行計畫。當最佳的執行計畫被選出後，便交給 Plan Executor 執行，最後將結果輸出。

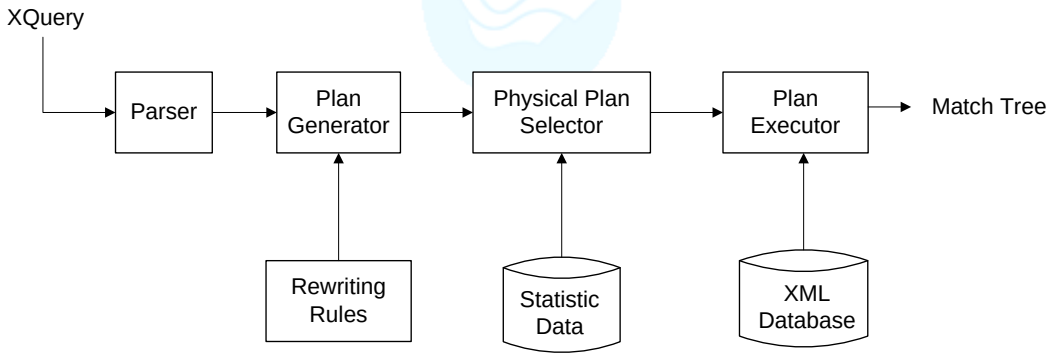

圖 4.1：系統架構圖

4.2 Operators 與 Cost Model

針對在第二章中提到的兩類作法，我們將其作法中可以被獨立分割出來的部分，定義成為一個個獨立的運算子 (Operator)，以支援系統的最佳化處理。在本節中，我們將會詳細介紹各個 Operator 的用途以及對應的 Cost。

在我們的系統中，根據 Operator 的性質以及所處理的資料類別， 我們將 Operator 分為三大類，第一類是針對資料擷取的 Operator，第二類則是針對資訊

檢索限制判斷，以及路徑限制判斷的 Operator，第三類則是處理最後結構判斷及整合的 Operator 。圖 4.2 為所有 Operator 的列表，其中除了列出 Operator 之外，我們同時也列出了各個 Operator 輸出的資料為何(對應第二章的定義)。

	Operator	Output
I/O	T(t)	T_match(t)
	K(k)	K_match(k)
Stream/SCU	$C_{op}^{t1,t2}(.)$	C_match(op)
	$P_{path}(.)$	P_match(p)
Twig Join	$TJoin_{QueryTree}(st\ldots*)$	Match Tree/Answer

圖 4.2：Logical Operators

針對各個 Operator 所對應的 Cost Model 以及其對應的演算法列表如下，另外，最後的兩個 Operator 是在實際運算時會使用到的 Operator：

Operator		Cost Model	Algorithm
T(t)		$C_1 * \|T(t)\|$	getTag
K(k)		$C_2 * \|K(k)\|$	getSCUTable
$C_{op}^{t1,t2}(.)$	Stream	$C_3 * \|Termlist\|$	TJIR_OrderHandle TJIR_DistanceHandle
	SCU	$C_4 * \|SCU\|$	SCU_OrderHandle SCU_DistanceHandle
$P_{path}(.)$		$C_5 * \|P_{path}(.)\|$	Pmatch
$TJoin_{QueryTree}$ (st…*)		$C_6 * \|I\| + C_7 * \|O\|$	TJoin
$LCA(SCU_1, SCU_2)$		$C_8 * (\|SCU_1\|+\|SCU_2\|) + C_9 * \| SCU_o \|$	DoingLCA
PostToPre(SCU)		$C_{10} * \|SCU\|$	PostToPreorderlist

圖 4.3：Physical Operator 及 Cost Model、Algorithm 對應表

圖 4.3 中的演算法 getTag、TJIR_OrderHandle、TJIR_DistanceHandle 請參見論文[吳 09]，演算法 getSCUTable、SCU_OrderHandle、SCU_distanceHandle、DoingLCA 請參見論文[ACD06]，演算法 TJoin、Pmatch 請參見論文[LLCC05]，演算法 PostToPreorderlist，則是先將 postorder 轉成 inorder 之後，再將 inorder 轉換為 preoder。

在上述定義的 Cost Model 中，針對處理 C_match 的部份，因為在處理時會遇到處理 Termlist 以及 SCU 兩種資料型態，所以在計算 Cost 的時候，會有不同的計算方式。而表格中的 I 代表輸入的資料量，O 代表此部份輸出的資料量。

在 Cost Model 中的 c1, c2, c3, c4, c5, c6, c7, c8, c9, c10 是該 Cost Model 的執行係數。由於在實際執行的環境中，各個演算法所花費的時間不同，因此我們針對不同的 operator 找出其對應的係數，各個係數列表如下：

Variable	Description
C_1, C_2	Average cost for I/O and internal representation
C_3	Cost for Termlist Constraint Compare
C_4	Cost for SCU Constraint Compare
C_5	Cost for Dewey ID converting
C_6	Cost for getting next node
C_7	Cost for merging path solution
C_8	Cost for LCA calculating
C_9	Cost for SCU handling
C_{10}	Cost for order converting

圖 4.4：Cost Model 係數對應表

我們根據各個演算法的 time complicity，列出對應的 cost model。以下做進一步的說明。針對取得標籤元素的 T(t)以及取得關鍵字元素的 K(k)，其執行時先是透過 block I/O，將符合標籤限制 t 或關鍵字限制 k 的所有元素從硬碟中取出，即為找出符合 T_match 或 K_match 的元素，並且逐一將取得的標籤內出現的關鍵字建立成為一組 Term List，而 K(k)則是將符合關鍵字的所有元素建立到一個 SCU table，最後並將所有的 Term List 或 SCU Table 掛回 Query Tree 當中。由於合併 Term List 或 SCU Table 的動作都是在記憶體中執行，並且其執行時間與讀取到的資料比數成正比，因此我們直接定義該部分的分數即為符合標籤限制 t 或關鍵字限制 k 的資料數量，也就是 Term List 或 SCU Table 的數量。而此部份的時間複雜度，也就是與輸入的資料量成正比關係，因此為 O(|t|) 或 O(|k|)。

而在資訊檢索限制C_{op}^{term}(st/SCU)的部份，在判斷是否符合限制時，所需要花掉的時間因為輸入資料結構的不同，我們分成兩個計算方式，第一種是處理 Termlist 的狀況，在此狀況下，Operator 所需要處理的資料為前一級輸出的 Termlist 結果，判斷每個 Termlist 內的資料是否符合條件限制驗，因此其時間複雜度為 O(|Termlist|)。第二種狀況則是遇到處理 SCU 的狀況，在此狀況下因為需要先對 SCU 做 LCA，接著才能開始判斷是否符合資訊檢索限制。處理 LCA 的部份我們稍後再討論。至於判斷是否符合資訊檢索限制的部份與 Termlist 的時間複雜度相同，唯一的差異只是輸入的資料型態為 SCU Table，所以時間複雜度為 O(|SCU|)。

判斷是否符合路徑限制P_{path}(st/SCU)時，其最關鍵的部份，就在於將 Term List 或 SCU table 中的 DeweyID 轉換回路徑，並且和 Query Tree 當中的路徑進行比對，因此此部份的分數計算，我們是將轉換的部分單獨找出其係數 c6，乘上此部份的輸出資料筆數得之。

在 TJoin 的部份，首先會先對輸入的資料進行結構的判斷，最後再將符合結構的結果整理合併輸出，因此此部份的時間複雜度必須考慮輸入的資料以及最後在整理合併輸出時的資料，其時間複雜度為 O(|Input| + |Output|)，其中 Input 的部份為前級輸入的 Termlist 的資料，而 Output 則是最後輸出的 Match Tree 中的資料。

而最後的兩個 Operator，LCA 以及 PostToPre 並未出現在圖 4.2 的表格中，但是因為這兩個 Operator 實際運算過程中會被使用到，其 Cost Model 必須被考慮，因此我們將他們列於圖 4.3 的 Physical Operator 表格中。其中的 LCA(SCU1, SCU2)是在做完 K_match 後，將兩個關鍵字整合成為一個 SCU 表格。因此每當 Query 中出現一個資訊檢索限制時，我們就必須計算一次 LCA，而每次計算時，因為需要處理兩個 K_match 取得的結果，並且在計算後的結果，我們必須要重新整理至一個新的 SCU Table 上，因此在時間複雜度上即為 O((|SCU1|+|SCU2|) + | SCUo|)。另外因為每次出現一個資訊檢索限制，我們就必須計算一次 LCA，

而每個資訊檢索限制計算完的結果，又還必須和前面一個計算的結果再做一次 LCA，因此在一個 Query 中總共計算 LCA 的次數為 2n-1，其中 n 為資訊檢索限制的數量。

而 PostToPre 則是當 Keyword-first 處理到最後，要將結果利用 TJoin 整合時，因為 TJoin 只能接受 Preorder 排序的資料，因此必須將原本 SCU 當中，利用 Postorder 排序的資料，轉換為 Preorder 排序。在轉換時，我們所需要處理的量，即為輸入的 SCU 資料量，因此時間複雜度即為 O(|SCU|)。

4.3 計算係數的方法

在前一節中的 Cost Model 中，我們有定義了許多的係數，透過這些係數，我們讓我們的 Cost Model 變得更能反應實際的執行效率。而在這節中，我們將會說明我們是如何找出這些係數。

由於我們主要考慮的因素是執行時間，因此找尋係數的方法也是由演算法中的執行時間，經過多次的執行測試，並且利用平均值取的得數值。

首先我們選定四組針對 DBLP 以及五組針對 XMark 的測試 Query (其中 DBLP 以及 XMark 的大小皆為 50MB)，並且將每個 Query 重複執行 20 次，每次執行時，針對我們定義的係數，它的執行時間與該部分處理到的資料量的比數都會被記錄下來，由此可得到一筆資料所需要花的時間，最後再將所有的記錄加總後取平均，即可得到所需要的係數。

$$Coefficient = \frac{\sum \frac{執行時間}{處理的資料量}}{總執行次數}$$

圖 4.5：係數計算公式

```
For $p in document (“http:://dblab.cs.ntou.edu.tw/dblp.xml”)
        /dblp/inproceedings
where $p/booktitle contains text ((“System” ftand “System”) ordered)
```

```
    $p/title contains text ((“program” ftand “program”) ordered)
return $p
```

圖 4.6：尋找係數 C1、C2 的測試 Query 1

```
For $p in document (“http:://dblab.cs.ntou.edu.tw/dblp.xml”)
        /dblp/inproceedings
where $p/booktitle contains text ((“System” ftand “System”) ordered)
    $p/title contains text ((“language” ftand “language”) ordered)
return $p
```

圖 4.7：尋找係數 C1、C2 的測試 Query 2

```
For $p in document (“http:://dblab.cs.ntou.edu.tw/xmark.xml”)
        /site//item
where $p/description//text contains text ((“master” and “master”) ordered)
    $p/mailbox//from contains text ((“Mehrdad” ftand “Mehrdad”) ordered)
return $p
```

圖 4.8：尋找係數 C1、C2 的測試 Query 3

```
For $p in document (“http:://dblab.cs.ntou.edu.tw/dblp.xml”)
        /dblp/inproceedings
where $p/booktitle contains text ((“System” ftand “System”) ordered) ftand
                ((“Advance” ftand “Course”) distance <= 2 words)
 $p/title contains text ((“Language” ftand “Language”) ordered)
return $p
```

圖 4.9：尋找係數 C3、C4 的測試 Query 4

```
For $p in document (“http:://dblab.cs.ntou.edu.tw/dblp.xml”)
        /dblp/inproceedings
where $p/booktitle contains text ((“System” ftand “System”) ordered) ftand
                ((“Advance” ftand “Course”) ordered)
        ftand ((“Advance” ftand “Course”) distance <= 2 words)
      $p/title contains text ((“Language” ftand “Language”) ordered)
```

```
return $p
```

圖 4.10：尋找係數 C3、C4 的測試 Query 5

```
For $p in document (“http:://dblab.cs.ntou.edu.tw/xmark.xml”)
        /site/regions/asia/item
where $p//emph contains text ((“master” ftand “attend”) distance <= 500 words )
      $p//bold contains text ((“ship” ftand “see”) ordered)
return $p
```

圖 4.11：尋找係數 C3 、C4 的測試 Query 6

```
For $p in document (“http:://dblab.cs.ntou.edu.tw/xmark.xml”)
        /site/regions/asia/item
where  $p//description//text contains text ((“master” ftand “attend”) distance >= 5
words )
        $p//shipping contains text ((“see” ftand “see”) ordered)
return $p
```

圖 4.12：尋找係數 C5、C6、C7 的測試 Query 7

```
For $p in document (“http:://dblab.cs.ntou.edu.tw/xmark.xml”)
        /site//item
where  $p/description//text contains text ((“master” ftand “master”) ordered)
        $p/mailbox//from contains text ((“Mehrdad” ftand “Mehrdad”) ordered)
return $p
```

圖 4.13：尋找係數 C5、C6、C7 的測試 Query 8

```
For $p in document (“http:://dblab.cs.ntou.edu.tw/xmark.xml”)
        /site//item
where $p//description//text contains text ((“master” ftand “master”) ordered)
        $p//description//keyword contains text ((“bound” ftand “bound”) ordered)
        $p//shipping contains text ((“description” ftand “description”) ordered)
return $p
```

圖 4.14：尋找係數 C5、C6、C7 的測試 Query 9

```
For $p in document (“http:://dblab.cs.ntou.edu.tw/xmark.xml”)
        /site//item
where $p//emph contains text ((“master” ftand “attend”) ordered)
        $p//bold contains text ((“ship” ftand “see”) ordered)
return $p
```

圖 4.15：尋找係數 C8、C9、C10 的測試 Query 10

```
For $p in document (“http:://dblab.cs.ntou.edu.tw/dblp.xml”)
        /dblp/inproceedings
where $p/booktitle contains text ((“Database” ftand “System”) ordered)
    $p/title contains text ((“query” ftand “optimization”) ordered)
return $p
```

圖 4.16：尋找係數 C8、C9、C10 的測試 Query 11

```
For $p in document (“http:://dblab.cs.ntou.edu.tw/dblp.xml”)
        /dblp/inproceedings
where $p/booktitle contains text ((“XML” ftand “Work”) ordered)
    $p/title contains text ((“data” ftand “application”) ordered)
return $p
```

圖 4.17：尋找係數 C8、C9、C10 的測試 Query 12

圖 4.6 到圖 4.17 為我們所選定的測試 Query。Query 1 到 Query 3 主要是用來測試 T_match 以及 K_match 的 I/O 係數 C_1 以及 C_2。這些 Query 在兩個 Where 子句的關鍵字限中，都各只有兩個相同的關鍵字，而且所使用的限制條件為 ordered，亦即只限制需內容有出現該關鍵字即可。

至於 Query 4 到 Query 6 則就是用來測試判斷資訊檢索限制的係數 C_3、C_4。這些 Query 在資訊檢索限制中，除了基本的 ordered 限制式外，還另外有兩個不同的關鍵字使用 distance 的限制，因此在執行資訊檢索限制的時間比較顯著。

Query 7 到 Query 9 我們用了 XMark 的資料進行測試，並且使用跳層的路徑限制，主要是用來測試針對 P_match 判斷時，在路徑轉換所用到的係數 C_5，同時也測試最後使用 TJoin 整合結果時，會用到的係數 C_6、C_7。

至於 Query 10 到 Query 12 則是在資訊檢索限制內，分別下了不同的關鍵字，以用來測試 SCU 在計算 LCA 的係數 C_8 和 C_9，同時也測試在將 Postorder 轉成 Preorder 時所會用到的係數 C_{10}。

Variable	Coefficient
C_1	12
C_2	15
C_3	1.56
C_4	1.33
C_5	4.24
C_6	1
C_7	4
C_8	1
C_9	5
C_{10}	0.1

圖 4.18：係數

由於我們一開始找到的原始係數都是在 10^{-3} 到 10^{-6} 之間，因此為了讓係數正規化，我們統一將所有係數都乘上 10^{+6}。圖 4.18 即為透過上述的 Query 以及圖 4.5 的計算公式，所找出的係數。我們可看到做 I/O 所花的時間較多，做 order 轉換最快，其餘的 cost 則大小類似。

接著，我們利用範例 4.1 來說明如何計算執行計畫的分數。

【範例 4.1】：

給一 Query，Q1 如下：

```
For $p in document (“http:://dblab.cs.ntou.edu.tw/dblp.xml”)
        /dblp/inproceedings
where $p/booktitle contains text ((“System” ftand “System”) ordered) ftand
                ((“Advance” ftand “Course”) distance <= 2 words)
      $p/title contains text ((“Language” ftand “Language” ordered)
return $p
```

其 Physical Algebra 表示式之一如下：

TJoin($P_{/dblp/inproceedings/booktitle}(C^{A,C}_{distance}(C^{S,S}_{order}(T(booktitle))))$,$P_{/dblp/inproceedings/title}(C^{L,L}_{order}(T(title)))$)

則計算該執行計畫分數的算法如下：

$$
\begin{aligned}
Cost = \; & C_1 * |T(booktitle)| + \\
& C_3 * |Termlist_{booktitle}| + \\
& C_3 * 0.5 * |Termlist_{booktitle}| + \\
& C_5 * |P_{/dblp/inproceedings/booktitle}(Termlist_{booktitle})| + \\
& C_1 * |T(title)| + \\
& C_3 * |Termlist_{title}| + \\
& C_5 * |P_{/dblp/inproceedings/title}(Termlist_{title})| + \\
& C_6 * |0.5*0.8*Termlist_{title}+0.5*Termlist_{booktitle}| + \\
& C_7 * |(0.5*0.8*Termlist_{title}+0.5*Termlist_{booktitle}) * 0.5|
\end{aligned}
$$

在範例 4.1 當中，針對所有的 ordered 限制和 distance 限制，以及 LCA 的 selectivity 假設為 0.5，而 P_match 如果是在使用結構優先的方法時，因為是取得符合結構限制中的標籤元素，因此在大多數的情況下都已經都是符合結構的限制，所以我們假設為 0.8，而在關鍵字優先時，則假設為 0.5。

4.4 Rewriting Rule

在 Cost-Based Optimization 中，除了計算各個執行計畫的分數之外，最重要的部份，就是如何將現有的執行計畫改寫。這是因為同一個 Query 可能可以擁有很多個不同的執行計畫，而不同的執行計畫會影響此 Query 的執行效率。本節將介紹我們所定義出來的 Rewriting Rule，以及如何套用 Rewriting Rule 的演算法。

Rewriting Rules:
1. $C_{op1}^{t1,t2}(C_{op2}^{t3,t4}(.)) = C_{op2}^{t3,t4}(C_{op1}^{t1,t2}(.))$ | Op_1 , Op_2 = Order 或 Distance 2. $P_{path}(C_{op}^{t1,t2}(.)) = C_{op}^{t1,t2}(P_{path}(.))$ 3. LCA($C_{op1}^{t1,t2}$(LCA(SCU_1,SCU_2)), $C_{op1}^{t3,t4}$(LCA(SCU_3,SCU_4))) = $C_{op2}^{t3,t4}$($C_{op1}^{t1,t2}$(LCA(LCA(SCU_1,SCU_2), LCA(SCU_3,SCU_4)))) 4. LCA($C_{op1}^{t1,t2}$(LCA(SCU_1,SCU_2)), $C_{op1}^{t3,t4}$(LCA(SCU_3,SCU_4))) = $C_{op2}^{t3,t4}$(LCA($C_{op1}^{t1,t2}$(LCA(SCU_1,SCU_2), LCA(SCU_3,SCU_4)))) 5. LCA($C_{op1}^{t1,t2}$(LCA(SCU_1,SCU_2)), $C_{op1}^{t3,t4}$(LCA(SCU_3,SCU_4))) = $C_{op2}^{t3,t4}$($C_{op1}^{t1,t2}$(LCA(LCA(LCA(SCU_1,SCU_2), SCU_3), SCU_4)))

圖 4.19：Rewriting Rules

圖 4.19 為所有的 Rewriting Rules 列表。其中 rule 1 和 rule 2 適用於輸入是 SCU Table 或 Termlist，而 rule 3-5 則是只針對 SCU Table，以下我們將逐一介紹：

Rule 1：

由於先判斷 Order 後再判斷 Distance 或是先判斷 Distance 後再判斷 Order，不會影響到輸出結果，因此此條 Rule 則是允許當連續執行兩個資訊檢索限制時，可以將兩個不同的限制執行順序對調 。

【範例 4.2】：

給一 Query，Q1 如下:

```
For $p in document ( “http:://dblab.cs.ntou.edu.tw/dblp.xml” )
        /dblp/inproceedings
where $p/booktitle contains text ((“System” ftand “System”) ordered) ftand
                ((“Advance” ftand “Course”) distance <= 2 words)
```

```
        $p/title contains text ((“Language” ftand “Language” ordered)
return $p
```

若其 Physical Algebra 表示式之一如下（每個關鍵字我們只表示其第一個英文字母）：

TJoin($P_{/dblp/inproceedings/booktitle}(C^{A,C}_{distance}(C^{S,S}_{order}(T(booktitle))))$,$P_{/dblp/inproceedings/title}(C^{L,L}_{order}$ ((T(title)))))

若套用 Rewriting Rule 1，則可改寫為：

TJoin($P_{/dblp/inproceedings/booktitle}(C^{S,S}_{order}(C^{A,C}_{distance}(T(booktitle))))$,$P_{/dblp/inproceedings/title}(C^{L,L}_{order}$ (T(title)))))

Rule 2：

另外在擷取完資料後，除了先執行資訊檢索限制外，亦可先行判斷是否符合路徑限制之後再判斷資訊檢索限制，因此 Rule 2 可將先執行資訊檢索限制判斷再執行路判斷改寫為，先判斷路徑之後再執行資訊檢索限制判斷。

【範例 4.3】：

延續範例 4.2 若再套用 Rewriting Rule 2 兩次，則可改寫為：

TJoin($C^{S,S}_{order}(C^{A,C}_{distance}(P_{/dblp/inproceedings/booktitle}(T(booktitle)))$, $P_{/dblp/inproceedings/title}(C^{L,L}_{order}$ (T(title)))))

而在另一種情況下，我們也可以將路徑判斷安插在已經拆解開的兩個資訊檢索限制判斷中間，也就是利用資訊檢索篩選掉的一部分結果，先進行路徑判斷，然後再接著完成剩下的資訊檢索判斷。例如，我們將範例 4.2 中的表示式，僅套用 rewriting rules 2 一次則會得到

TJoin($C^{A,C}_{distance}$($P_{/dblp/inproceedings/booktitle}(C^{S,S}_{order}(T(booktitle))))$,$P_{/dblp/inproceedings/title}(C^{L,L}_{order}$ ((T(title)))))

Rule 3：

在使用關鍵字優先的方法時，在判斷資訊檢索限制前，需要先取得資訊檢索限制內的關鍵字，而在取得關鍵字後，需要針對取得的關鍵字計算 LCA，而當出現兩個資訊檢索限制時，就必須在做完第一個資訊檢索判斷後，先針對第

二組限制的關鍵字做 LCA，再執行第二個資訊檢索限制。但是我們也可以先將兩個資訊檢索內的關鍵字取得後，先針對這些關鍵字計算 LCA，然後再開始判斷資訊檢索限制。

【範例 4.4】：

我們將範例 4.2 的查詢句 Q1 套用關鍵字優先的 Physical Algebra 表示式如下：

TJoin(PostToPre(P/dblp/inproceedings/booktitle(LCA($C^{S,S}_{order}$(LCA(K(System),K(system))), $C^{A,C}_{distance}$(LCA(K(Advance),K(Course))))),
PostToPre(P/dblp/inproceedings/title($C^{L,L}_{order}$(LCA(K(Language),K(Language))))))))

套用 Rewriting Rule 3 之後則為：

TJoin(PostToPre(P/dblp/inproceedings/booktitle($C^{S,S}_{order}$($C^{A,C}_{distance}$(LCA(LCA(K(System),K(system)), LCA(K(Advance), K(Course))))),
PostToPre(P/dblp/inproceedings/title($C^{L,L}_{order}$(LCA(K(Language),K(Language))))))))

若再將結果分別套用 Rewriting Rule 2 一次及兩次，則會得到：

- TJoin(PostToPre($C^{A,C}_{distance}$(P/dblp/inproceedings/booktitle($C^{S,S}_{order}$(LCA(LCA(K(System),K(system)), LCA(K(Advance), K(Course))))),
  PostToPre(P/dblp/inproceedings/title($C^{L,L}_{order}$(LCA(K(Language),K(Language))))))))
- TJoin(PostToPre($C^{S,S}_{order}$($C^{A,C}_{distance}$(P/dblp/inproceedings/booktitle(LCA(LCA(K(System),K(system)), LCA(K(Advance), K(Course))))),
  PostToPre(P/dblp/inproceedings/title($C^{L,L}_{order}$ (LCA(K(Language),K(Language))))))))

4.5 產生等價執行計畫

在本節中，我們將介紹如何套用 Rewriting Rules，並且如何產生所有可能的執行計畫，以提供 Optimizer 選擇最佳的執行計畫。

要能順利找出最佳的執行計畫，我們必須先將所有套用 Rewriting Rules 可能產生的結果列出。由於我們的 Rewriting Rules 都只有在 QueryTree 的葉節點(Leaf Node)中套用，因此我們先將 QueryTree 輸入，並且針對每個葉節點各執

行一次 Rewriting Rule 套用。為了處理上的方便，我們將各個 Operator 利用單一英文字母代替，$C_{Distance}^{t1,t2}$由字母 “D” 代替，$C_{Order}^{t1,t2}$由字母 “O” 代替，P_{path}由字母 “P” 代替，T(t)由字母 “T” 代替，K(k)由字母 “K” 代替，LCA 由字母“L”代替。在圖 4.20 中，L03 開始到 L13 之間，首先產生預設值行計畫，並將預設的執行計畫丟入計畫池(Pool)中，接著在 L14 到 L24 之間開始套用 Rewriting Rules。預設執行計畫我們參考[吳 09]中，針對結構優先以及關鍵字優先所設計的系統執行順序。執行的順序為：order 先做，接著做 distance，然後再做 Pmatch，其中若是使用關鍵字優先的方法，則會在做 order 前，先做 LCA。至於選擇先做 order 再做 distance 的原因則是，處理 distance 限制需要額外對 position 做運算，所花費的時間較多，因此當兩種限制都存在時，會先處理 order 限制，使得交給 distance 的資料量減少。至於當遇到資訊檢索限制當中出現不只一個 order 或 distance 限制時，我們在實作上，將所有的相同資訊檢索限制的運算子都緊接著一起做，因此在計畫的表示上，都只用一個符號表示，不會重複出現。

在 L16 首先先將前面已經產生的執行計畫套用前面可用的 Rewriting Rules，並且將套用的結果收集在集合 AppliedPlans 內。但是因為有可能會產生已經存在過的執行計畫，因此在 L17 的地方先判斷 AppliedPlans 是否完全包含於現有的計畫池中。如果並未完全包含，即代表套用 Rewriting Rules 之後，仍有新產生的執行計畫，因此在 L08 先將 AppliedPlans 當中已經出現在計畫池內的執行計畫扣除，接著在 L20 把新的執行計畫加入執行計畫池內。然而，如果在 L16 套用 Rewriting Rules 之後產生的計畫，都已經有出現在計畫池內，那就代表已經沒有其他的 Rewriting Rules 可以套用了。此時即可離開，並且將所有的執行計畫回存到節點中。

演算法名稱：PlanGenerator 輸入：QueryTree 輸出：Set of Plans	
L01	Pool = ∅
L02	Foreach(Leaf node N in QueryTree)
L03	{
L04	Plan = Plan + “P”;
L05	If(OrderedConstraint != NULL)
L06	Plan = Plan + “O”;

```
L07       If(DistanceConstraint != NULL)
L08             Plan = Plan + “D”;
L09       PlanK = Plan + “KK”
L10       Pool = Pool ∪ PlanK;
L11       PlanT = Plan + “T”
L12       Pool = Pool ∪ PlanT;
L13       Plans = Pool;
L14       While(Plans != ∅)
L15       {
L16             AppliedPlans = ApplyRules(Plans);
L17             If(AppliedPlans ⊄ Pool)
L18             {
L19                   Plans = AppliedPlans – Pool;
L20                   Pool = Pool ∪ Plans;
L21             }else{
L22                   Plans = ∅;
L23             }
L24       }
L25       SavePoolToNode();
L26 }
```

圖 4.20：產生所有可能執行計畫演算法

接著我們來說明在圖 4.20 中 L16 的 ApplyRules 是如何套用 Rewriting Rule。如圖 4.21 所示，因為我們已經將各個 Operator 轉換為單一的英文字母表示，因此我們同時也可以將 Rewriting Rules 用同樣的方法表示，並且儲存在一個二維陣列 Rule[m][n]中，m 代表是第幾個 Rule，n 代表的是此 Rule 中左式的第 n 個 Operator， 另外針對輸入的執行計畫，我們也同樣利用陣列 P[n] 表示，n 代表的是此計畫中的第 n 個 Operator。接著我們針對 P 當中的每個 Operator 逐一走訪，每次走訪時，都將目前拜訪到的 Operator 先放到堆疊 S 當中(L08)，接著去判斷目前堆疊中的第 n 個是否有與第一條 Rule 當中的第一個 Operator 相同，並且在堆疊中的第 n-1 個是否與第一條 Rule 當中的第二個 Operator 相同。如果符合上述條件的話，則在 L12 將 P 當中的這兩個 Operator 交換，即可符合該 Rewriting Rule 的轉換結果。在轉換好執行計畫後，還必須將在堆疊中剛剛已經轉換過的 Operator 給 Pop 出來。但是因為目前到訪的 Operator 可能還可以與前面的 Operator 進行轉換，因此我們需將目前到訪的 Operator 再重新放回堆疊中(L14)。最後我們就可將轉換好的結果回傳。

演算法名稱：ApplyRules

輸入：Set of Plans
輸出：Set of Plans

```
L01 NewPlans = ∅
L02 Foreach(P in Plans)
L03 {
L04 While(P[i] != '\n')
L05 {
L06         Push(S, P[i]);
L07         for(j=0;j<=1;j++)
L08         {
L09               Sn = Pop(S);    // S 為 Stack
L10               If(Rule[0][j] == S.top && Rule[1][j] == Sn)
L11               {
L12                     P = Exchange(P[i-1], P[i]);
L13                     Pop(S);
L14                     Pop(S);
L15                     Push(S, P[i]);
L16                     NewPlans = NewPlans ∪ P;
L17               }
L18               Else
L19                     Push(S, Sn);
L20         }
L21         i++;
L22 }
L23 }
L24 Return NewPlans;
```

圖 4.21：套用 Rewriting Rule 演算法

【範例 4.4】：

考慮範例 4.1 中的查詢句 Q1，透過前述的演算法，我們首先對此 Query 的 booktitle 葉節點產生出預設執行計畫，經過圖 4.20 的 L03 到 L13 產生的預設執行計畫為：PDOT 及 PLDLKKOLKK。其代表的執行順序為右到左，而其中 PDOT 為 Structure-first 的預設執行計畫，PLDLKKOLKK 則為 Keyword-first 的預設執行計畫。接著再將產生的預設執行計畫先放入計畫池內。然後在 L14 開始判斷目前是否有執行計畫可以套用 Rewriting Rule。

首先在圖 4.21 的 L02 開始，我們逐一針對輸入的執行計畫進行走訪。我們首先先針對執行計畫 PDOT 處理。在第一次迴圈時，我們先將第一個 Operator P 放入堆疊中，接著因為在圖 4.21 中的 L09 條件不符合，所以我們將

剛剛取出的 Operator 重新放回堆疊中並且繼續走訪下一個 Operator。在第二次迴圈時，因為此時堆疊中已經有兩個 Operator，P 及 D，而且在 L09 的判斷中符合第二條 Rule 的 Rule[0][1]中的 P 以及 Rule[1][1]中的 C，所以可以將這兩個 Operator 交換，並且將目前堆疊中的前兩個元素移除，另外也將改寫好的執行計畫丟入改寫計畫池內。當進入第三次迴圈時，此時的堆疊中取出的 Operator 為 D，而目前讀到的 Operator 為 O，兩者皆屬於 C_match，符合 Rule[0][0]中的 C 以及 Rule[1][0]的 C，因此可以將目前讀到的兩個 Operator 互換並且一樣將改寫後的計畫丟入改寫計畫池內。而在第四次迴圈時，因為已經沒有其他的 Operator 以及可以套用的 Rewriting Rule，所以便可離開迴圈並且回傳改寫後的計畫 NewPlans。

在此階段的 NewPlans 回傳的結果為：

1. DPOT
2. PODT

回到圖 4.20，剛剛回傳的執行計畫在 L17 判斷後，並未全部包含在執行計畫池內，因此進入 L19 將原本已經包含在計畫池內的執行計畫扣除後，再將結果放入執行計畫池內。

接著當圖 4.20 的 L14 迴圈再次執行的時候，我們就將剛剛產生的的兩個新執行計畫再次送到 ApplyRule 內去套用可用的 Rewriting Rule。

在圖 4.20 第二次執行 while 迴圈處理前面我們產生的第一個執行計畫 DPOT 時，將其套用 Rewriting Rule 會產生的執行計畫如下：

1. PDOT
2. DOPT

其中因為 PDOT 已經存在於執行計畫池內，所以在圖 4.20 的 L16 時就會被剃除掉，只剩下第二條執行計畫。接著，再繼續針對第二次迴圈所產生的第二條執行計畫 PODT 一樣套用 Rewriting Rule，其產生的結果如下：

1. OPDT
2. PDOT

然而第二條執行計畫 PDOT 在執行計畫池內也已經有了，因此在 L16 也會把它替除。最後在這趟的迴圈裡面，我們得到了 DOPT 以及 OPDT 兩個執行計畫。

接著再第三次的迴圈中，我們分別處理前面產生的兩個執行計畫，其中 DOPT 套用 Rewriting Rule 產生的計畫如下：

1. ODPT
2. DPOT

產生的結果中的第二條，因為已經出現在執行計畫池內，因此也一樣會被剃除。而前一次產生的執行計畫中的 OPDT 計畫，套用 Rewriting Rule 之後，所產生的執行計畫如下：

1. PODT
2. ODPT

在這邊可以看到，所產生的兩個執行計畫都已經有出現在執行計畫池內，因此全部都會被剃除。

最後，針對 Structure-first 的預設值行計畫 PDOT 套用 Rewriting Rule 之後的執行計畫為：

1. PDOT
2. DPOT
3. PODT
4. DOPT
5. OPDT
6. ODPT

而針對 Keyword-first 的預設值行計畫 PDOKK，則是與上述 Structure-first 相同的方式套用 Rewriting Rule。其產生的執行計畫如下：

1. PDOLLKKLKK
2. DPOLLKKLKK
3. LDLKKPOLKK
4. PODLLKKLKK
5. PLOLKKDLKK
6. DOPLLKKLKK
7. OPDLLKKLKK

8. LOLKKPDLKK
9. ODPLLKKLKK

針對查詢句 Q1 的第二個葉節點，我們同樣會針對 Structure-first 產生兩條計畫，以及針對 Keyword-first 產生兩條計畫。分別為：

Structure-first:

1. POT
2. OPT

Keyword-first:

1. POLKK
2. OPLKK

綜合以上的結果，我們共會替 Structure-first 產生 12 種計畫，以及對 Keyword-first 產生 18 種計畫。我們將會利用 4.2 節討論的 cost model 及 4.3 節所提出的係數，實地計算每個 plan 的 cost，再從中選出 cost 最低的 plan。以 Q1 為例，我們會選出的 plan 為 ODPKKOPKK。

第五章 實驗

在本章中，我們將設計數個不同的 XQuery，來評估我們所設計的系統效能。在實驗中，我們會將所有的執行計畫都產生，然後記錄每個 Query 執行時的最佳執行計畫時間、系統選擇的執行計畫執行時間、預設執行計畫的執行時間分做 SF(結構優先)和 KF(關鍵字優先)、最差的執行計畫執行時間以及選擇最佳執行計畫的執行時間，另外，我們也會列出系統所選出的最佳執行計畫、實際最佳執行計畫以及最差的執行計畫。

我們進行的實驗環境是使用時脈為 3.7GHz 的 Intel Core i7 CPU，以及記憶體為 16GB 的個人電腦，並且使用 Microsoft Windows 7 Enterprise 作業系統。至於系統的實作工具上，我們使用 Microsoft Visual C++ 2008 來實作，另外針對資料的索引建立，我們使用 Oracle Berkeley DB 來建立以及儲存索引資料。

針對所要進行的實驗，我們使用了兩種不同的測試資料，分別為德國 Universität Trier 建立的 DBLP(Digital Bibliography & Library Project)資料庫，以及常被使用來評估資料庫效能的 Xmark 資料庫。在 DBLP 資料庫中，主要是記錄從 1980 年代開始的計算機會議所發表的會議或期刊文章，而在我們的實驗中，我們使用大小為 107MB 的 DBLP XML 文件來進行實驗，其內容的年份是從 1980 年至 2003 年，另外，我們也使用了 DBLP 872MB 的 XML 文件，其內容年份則是從 1980 年至 2011 年。而 Xmark 則是記錄了許多物品的運籌資訊，我們利用 Xmark 所提供的資料產生器產生大小為 116MB XML 文件來進行實驗。

而我們所建立的索引資料當中，針對 DBLP 107MB 所建立出來的索引檔案大小，結構優先為 343MB，關鍵字優先為 67.3MB，針對 DBLP 872MB 所建立的索引檔案大小，結構優先為 1.45GB，關鍵字優先為 750MB，針對 Xmark 116MB 所建立的索引檔案大小，結構優先為 800MB，關鍵字優先為 123MB。

由於我們選用的兩種作法：結構優先、關鍵字優先，各具備不同的特質，因此我們在實驗的設計上，選擇了數種不同類別的 XQuery 來進行實驗，分別

為：遞增 IR 限制、關鍵字頻率、不同高度節點限制、標籤頻率、相同節點不同距離限制、遞增路徑限制。

在遞增 IR 限制中，我們逐一增加每個 Query Node 中的資訊檢索限制，從原本只有一個資訊檢索限制，增加到五個資訊檢索資訊檢索限制。而不同關鍵字頻率的實驗，我們則是將 DBLP 文件中統計得到的關鍵字數量分為低、中、高三類，並且利用資訊檢索限制，分別在 Query 中針對這三類的關鍵字進行查詢。不同標籤頻率的實驗，我們分別將 DBLP 以及 Xmark 的文件中統計出來的標籤個數一樣分成低、中、高三種，在路徑限制中，將查詢樹的葉節點限制在這三種類別中來進行實驗。在相同節點不同距離限制的實驗中，我們固定欲查詢的標籤以及關鍵字，但是調整資訊檢索限制中的距離限制。最後一個實驗，我們利用增加查詢樹中的葉節點數量，也就是增加 where 的限制式來進行實驗。

圖 5.1 為我們實驗所用到的 Query 總表，在這邊為了節省空間，我們的 XQuery 表示方式與前面的有些不同，在中括號之前的路徑即代表 For 子句以及 Return 子句中路徑，而中括號內則是 Where 子句的限制，其中包括資訊檢索限制。圖 5.2 及圖 5.3 為所有實驗的數據圖，其中 DBLP 中，有 90%我們選出的最佳執行計畫，就是實際的最佳執行計畫，其餘的執行時間差異也在 8%之內，而對 XMark，則是有達到 100%的最佳執行計畫，為實際的最佳執行計畫，顯示我們系統的有效性。由於 DBLP 兩個大小的 dataset 表現一致，所以在後面的討論中我們僅針對 107MB 的實驗數據加以討論。接下來，我們在各小節中做進一步的討論，而在每節當中，我們仍會將屬於該類別的實驗數據，繪製於該節當中。

Dataset	No	Query Statement	附註
遞增 IR 限制			
DBLP	Q1	/dblp/inproceedings[./title contains text (“system” ftand “system” ordered) ftand (“Advance” ftand “Course” ordered) ftand (“Algorithm” ftand “Application” ordered) ftand (“base” ftand “computer” ordered) ftand (“information” ftand “image” ordered) and ./booktitle contains text (“Language” ftand “Language” ordered) ftand (“Conference” ftand “Conference” ordered) ftand (“Data” ftand “performance” ordered) ftand (“Model”	10 個 IR 限制

		ftand “management” ordered) ftand (“design” ftand “design” ordered)]	
DBLP	Q2	/dblp/inproceedings[./title contains text (“system” ftand “system” ordered) ftand (“Advance” ftand “Course” ordered) ftand (“Algorithm” ftand “Application” ordered) ftand (“base” ftand “computer” ordered) ftand (“information” ftand “image” ordered) ftand (“computer” ftand “computer” ordered) ftand (“software” ftand “software” ordered) and./booktitle contains text (“Language” ftand “Language” ordered) ftand (“Conference” ftand “Conference” ordered) ftand (“Data” ftand “performance” ordered) ftand (“Model” ftand “management” ordered) ftand (“design” ftand “design” ordered) ftand (“Application” ftand “Application” ordered) ftand (“control” ftand “control” ordered)]	14 個 IR 限制
DBLP	Q3	/dblp/inproceedings[./title contains text (“system” ftand “system” ordered) ftand (“Advance” ftand “Course” ordered) ftand (“Algorithm” ftand “Application” ordered) ftand (“base” ftand “computer” ordered) ftand (“information” ftand “image” ordered) ftand (“computer” ftand “computer” ordered) ftand (“software” ftand “software” ordered) ftand (“parallel” ftand “parallel” ordered) ftand (“Distribute” ftand “Distribute” ordered) and. /booktitle contains text (“Language” ftand “Language” ordered) ftand (“Conference” ftand “Conference” ordered) ftand (“Data” ftand “performance” ordered) ftand (“Model” ftand “management” ordered) ftand (“design” ftand “design” ordered) ftand (“Application” ftand “Application” ordered) ftand (“control” ftand “control” ordered) ftand (“database” ftand “database” ordered) ftand (“ICRA” ftand “ICRA” ordered)]	18 個 IR 限制
DBLP	Q4	/dblp/inproceedings[./title contains text (“system” and “system” ordered) ftand (“Advance” ftand “Course” ordered) ftand (“Algorithm” ftand “Application” ordered) ftand (“base” ftand “computer” ordered) ftand (“information” ftand “image” ordered) ftand (“computer” ftand “computer” ordered) ftand (“software” ftand “software” ordered) ftand (“parallel” ftand “parallel” ordered) ftand (“Distribute” ftand “Distribute” ordered) ftand (“international” ftand “international” ordered) ftand (“logic” ftand “logic” ordered) and. /booktitle contains text (“Language” ftand “Language” ordered) ftand (“Conference” ftand “Conference” ordered) ftand (“Data” ftand “performance” ordered) ftand (“Model” ftand “management” ordered) ftand (“design” ftand “design” ordered) ftand (“Application” ftand “Application” ordered) ftand (“control” ftand “control” ordered) ftand (“database” ftand “database” ordered) ftand (“ICRA” ftand “ICRA” ordered) ftand (“John” ftand “John” ordered) ftand (“learn” ftand “learn” ordered)]	22 個 IR 限制
DBLP	Q5	/dblp/inproceedings[/title contains text (“system” ftand “system” ordered) ftand (“Advance” ftand “Course” ordered) ftand (“Algorithm” ftand “Application” ordered) ftand (“base” ftand “computer” ordered) ftand (“information” ftand “image” ordered) ftand (“computer” ftand “computer” ordered) ftand (“software” ftand “software” ordered) ftand (“parallel” ftand	26 個 IR 限制

		“parallel” ordered) ftand (“Distribute” ftand “Distribute” ordered) ftand (“international” ftand “international” ordered) ftand (“logic” ftand “logic” ordered) ftand (“network” ftand “network” ordered) ftand (“networks” ftand “networks” ordered) ftand /booktitle contains text (“Language” ftand “Language” ordered) ftand (“Conference” ftand “Conference” ordered) ftand (“Data” ftand “performance” ordered) ftand (“Model” ftand “management” ordered) ftand (“design” ftand “design” ordered) ftand (“Application” ftand “Application” ordered) ftand (“control” ftand “control” ordered) ftand (“database” ftand “database” ordered) ftand (“ICRA” ftand “ICRA” ordered) ftand (“John” ftand “John” ordered) ftand (“learn” ftand “learn” ordered) ftand (“network” ftand “network” ordered) ftand (“networks” ftand “networks” ordered)]	
關鍵字頻率			
DBLP	Q6	/dblp/inproceedings[./booktitle contains text (“Air” ftand “Air” ordered) and ./title contains text (“Consider” ftand “Consider” ordered)]	Low keyword frequency
DBLP	Q7	/dblp/inproceedings[./booktitle contains text (“Architecture” ftand “Architecture” ordered) and ./title contains text (“knowledge” ftand “knowledge” ordered)]	Medium keyword frequency
DBLP	Q8	/dblp/inproceedings[./booktitle contains text (“System” ftand “System” ordered) and ./title contains text (“Us” ftand “Us” ordered)]	High keyword frequency
標籤頻率			
DBLP	Q9	/dblp/inproceedings[./school contains text (“University” ftand “University” ordered) and ./note contains text (“Conference” ftand “Conference” ordered)]	Low tag frequency
DBLP	Q10	/dblp/inproceedings[./editor contains text (“John” ftand “Thomas” ordered) and ./publisher contains text (“Germany” ftand “Germany” ordered)]	Medium tag frequency
DBLP	Q11	/dblp/inproceedings[./title contains text (“System” ftand “System” ordered) and ./author contains text (“Jerry” ftand “Jerry” ordered)]	High tag frequency
不同高度節點限制			
XMark	Q12	/site/regions/asia/item[.//description//text//keyword contains text (“master” ftand “attend” distance at most 500 words) and .//shipping contains text (“ship” ftand “see” ordered)]	
XMark	Q13	/site/regions/asia/item[.//description//text contains text (“master” ftand “attend” distance at most 500 words) and .//shipping contains text (“ship” ftand “see” ordered)]	
XMark	Q14	/site/regions/asia/item[.//description contains text (“master” ftand “attend” distance at most 500 words) and .//shipping contains text (“ship” ftand “see” ordered)]	
同節點不同距離限制			
XMark	Q15	/site/regions/asia/item[.//desciption//text contains text (“master” ftand “attend” distance at least 5 words) and .//shipping contains text (“see” ftand “see” ordered)]	
XMark	Q16	/site/regions/asia/item[.//desciption//text contains text (“master” ftand “attend” distance at least 50000 words) and .//shipping contains text (“see” ftand “see” ordered)]	

XMark	Q17	/site/regions/asia/item[.//desciption//text contains text (“master” ftand “attend” distance at least 50000000000 words) and .//shipping contains text (“see” ftand “see” ordered)]	
遞增的路徑限制			
XMark	Q18	/site//item[.//description//text contains text (“master” ftand “master” ordered) and .//keyword contains text (“bound” ftand “bound” ordered) and .//shipping contains text (“description” ftand “description” ordered)]	
XMark	Q19	/site//item[.//description//text contains text (“master” ftand “master” ordered) and .//keyword contains text (“bound” ftand “bound” ordered) and .//location contains text (“Netherland” ftand “Netherland” ordered) and .//shipping contains text (“description” ftand “description” ordered)]	
XMark	Q20	/site//item[.//description//text contains text (“master” ftand “master” ordered) and .//keyword contains text (“bound” ftand “bound” ordered) and .//location contains text (“Netherland” ftand “Netherland” ordered) and .//shipping contains text (“description” ftand “description” ordered) and .//from contains text(“june” ftand “june” ordered)]	

表 5.1：實驗所用到的 Query

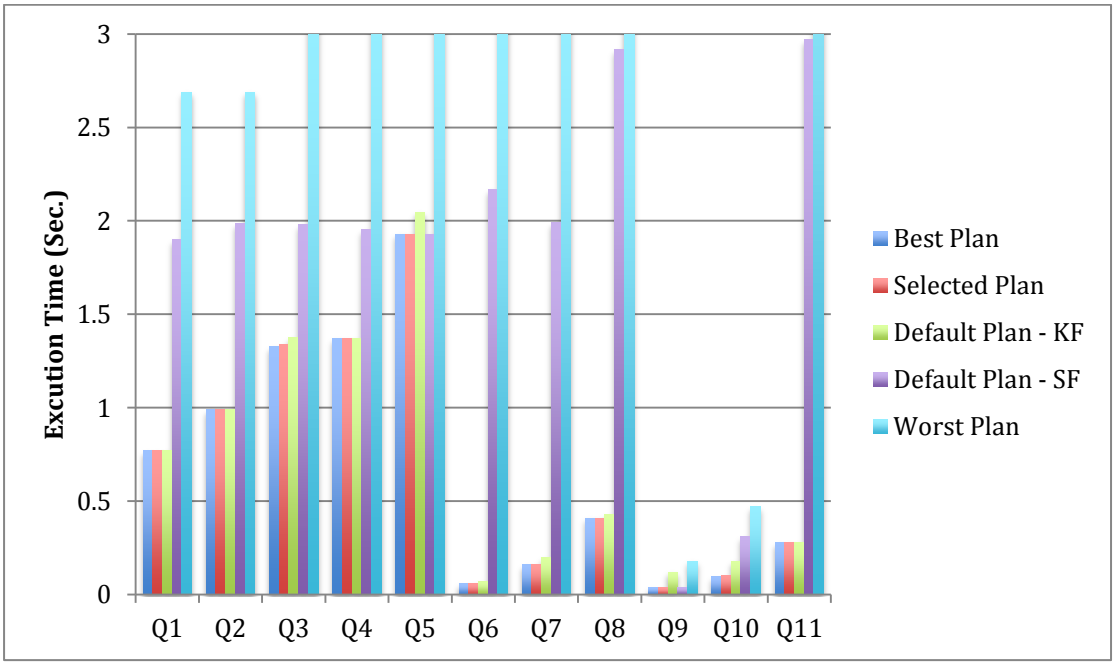

圖 5.1：DBLP Dataset 實驗數據 (107MB)

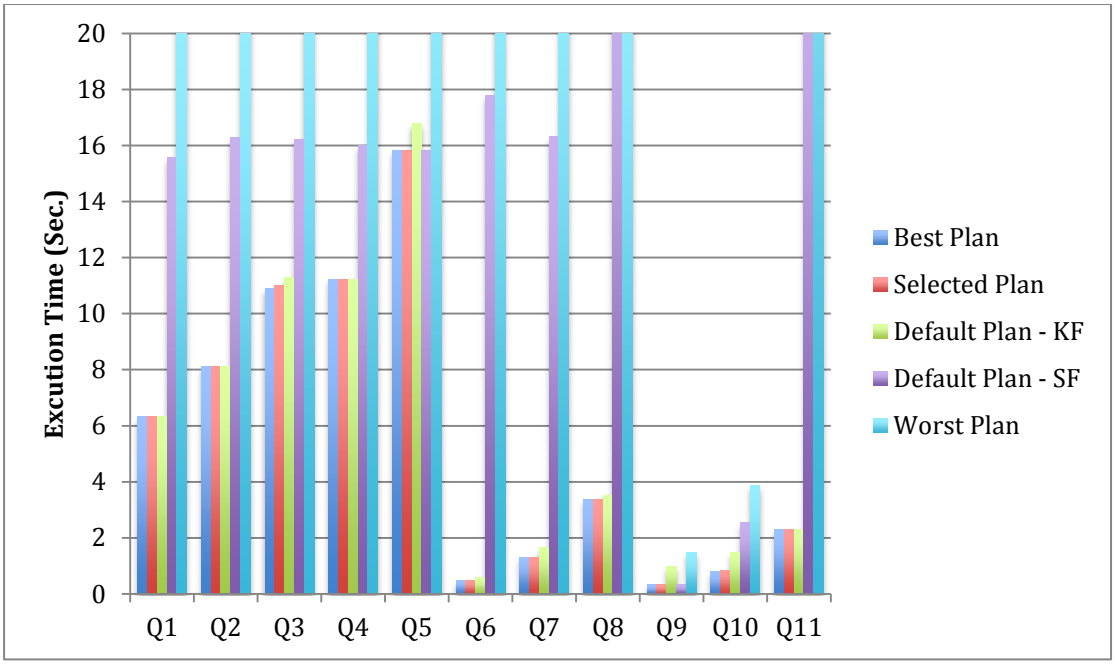

圖 5.2：DBLP Dataset 實驗數據 (872MB)

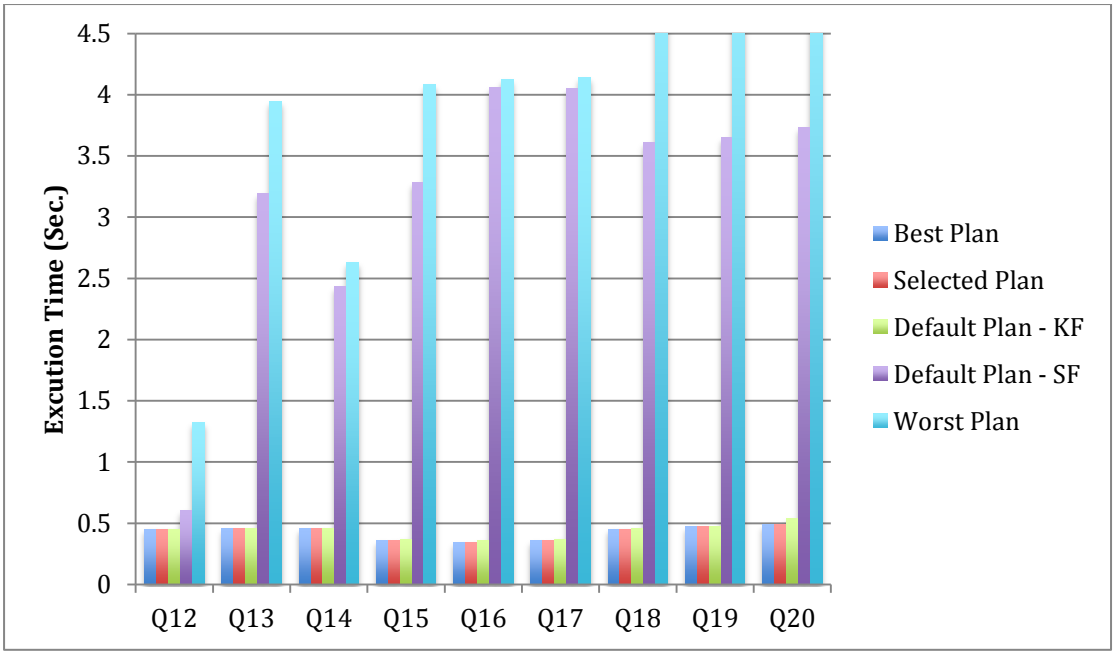

圖 5.3：XMark Dataset 實驗數據

5.1 遞增 IR 限制

在 Q1、Q2、Q3、Q4、Q5 當中，我們固定查詢樹中的葉節點，但是逐一增加葉節點當中的資訊檢索限制數量，為了便於觀察改變的狀況，此處都只採用 ordered 限制。在增加資訊檢索限制時，可以產生的改寫計畫也就相對增加，因此在產生可能的執行計畫所花的時間也就會稍微增加，但是如圖 5.4 所示，我

們將選擇最佳計畫的時間除以最佳的執行時間，其比率都在 4.5%以下。另外，因為對於關鍵字優先(KF)的作法而言，每增加一個資訊檢索限制，就必須增加取得關鍵字並且建立 SCU Table，以及將這些 SCU Table 計算 LCA 的次數，但是對於結構優先(SF)的方法而言，我們所下的標籤限制並沒有改變。所以 KF 做法的時間隨著限制增加而遞增，而 SF 的時間並沒有明顯的變化，所以最佳的執行計畫也隨著資訊檢索限制的增加，從原本選用關鍵字優先的作法，改到了結構優先的作法。

	Best Plan	Selected Plan	Default Plan - KF	Default Plan - SF	Worst Plan	PSA
Q1 執行時間	0.77	0.77	0.77	1.9	3.03	0.03
Q2 執行時間	0.99	0.99	0.99	1.984	3.07	0.04
Q3 執行時間	1.33	1.34	1.378	1.979	3.09	0.05
Q4 執行時間	1.37	1.37	1.37	1.952	3.123	0.04
Q5 執行時間	1.929	1.929	2.047	1.929	3.2	0.05

表 5.2：遞增 IR 限制之執行時間(Sec.)

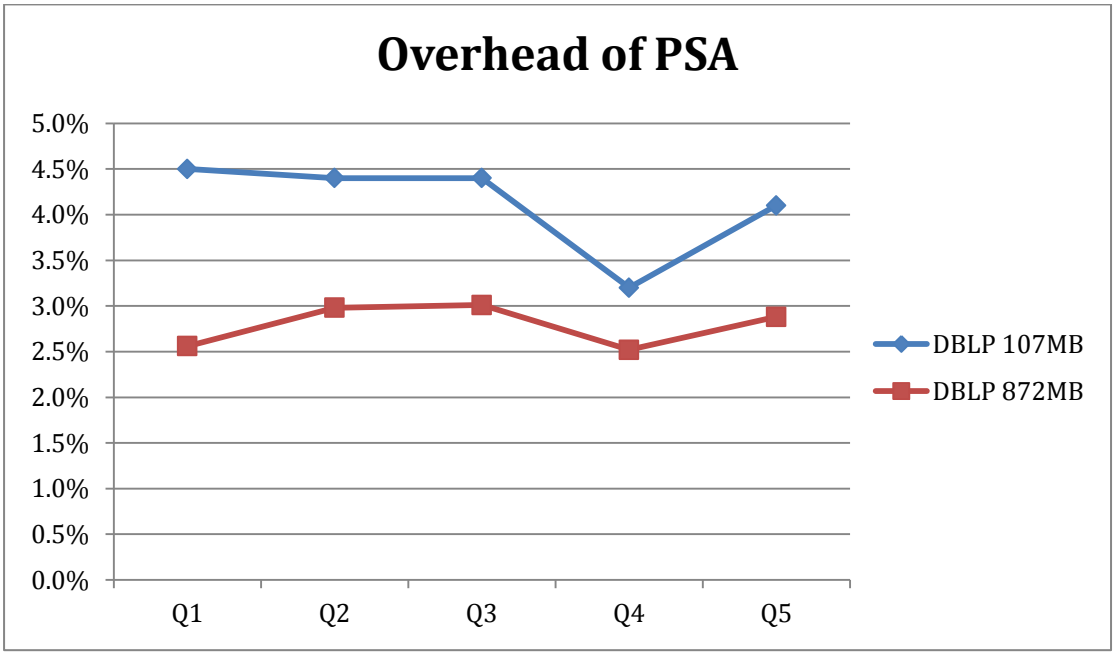

圖 5.4：PSA 與 Selected Plan 執行時間比

	Best Plan	Selected Plan	Worst Plan
Q1	JPOLLKKLLLKKLKKLL KKLKK POLLKKLLLKKLKKLL KKLKK	JPOLLKKLLLKKLKKLL KKLKK POLLKKLLLKKLKKLL KKLKK	JOPTOPT
Q2	JPOLLKKLLKKLLKKLL LKKLKKLLKKLKKPOL LKKLLKKLLKKLLLKK LKKLLKKLKK	JPOLLKKLLKKLLKKLL LKKLKKLLKKLKKPOL LKKLLKKLLKKLLLKK LKKLLKKLKK	JOPTOPT
Q3	JPOLLKKLLKKLLKKLL LKKLKKLLKKLKKPOL LKKLLKKLLKKLLLKK LKKLLKKLKK	JPOLLKKLLKKLLKKLL LKKLKKLLKKLKKPOL LKKLLKKLLKKLLLKK LKKLLKKLKK	JOPTOPT
Q4	JPOLLKKLLKKLLKKLL LKKLKKLLKKLKKPOL LKKLLKKLLKKLLLKK LKKLLKKLKK	JPOLLKKLLKKLLKKLL LKKLKKLLKKLKKPOL LKKLLKKLLKKLLLKK LKKLLKKLKK	JOPTOPT
Q5	JPOTPOT	JPOTPOT	JOPTOPT

表 5.3：遞增 IR 限制之最佳及最差執行計畫

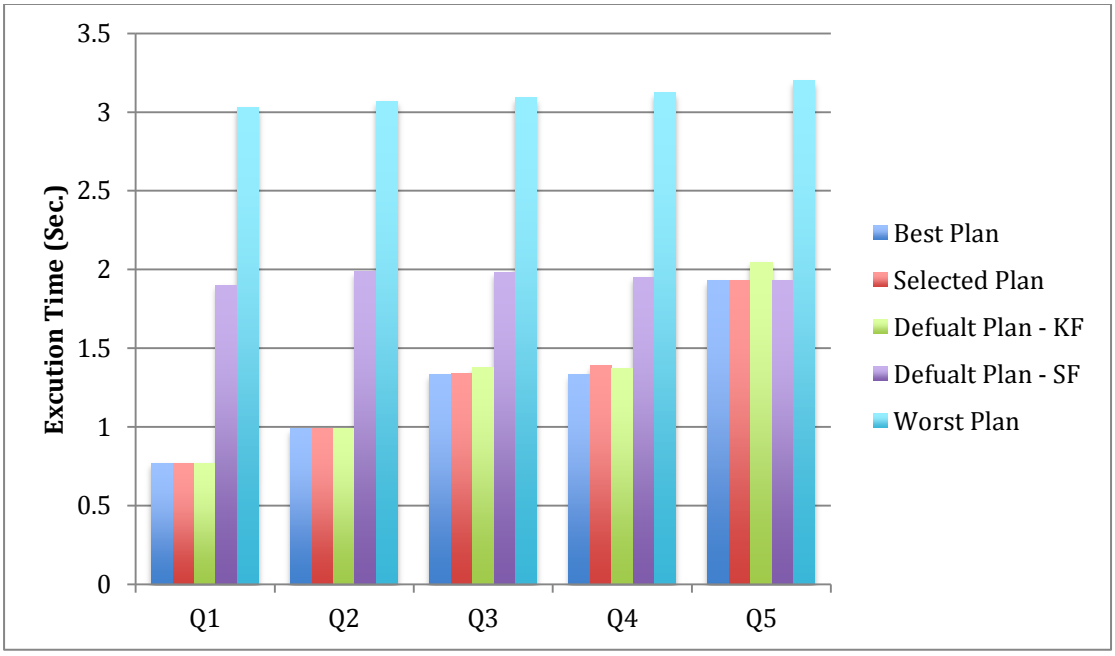

圖 5.5：遞增 IR 限制之整體執行時間

5.2 關鍵字頻率

接著我們利用 Q6、Q7、Q8 來實驗在不同關鍵字頻率下的執行效率以及 Cost Model 選擇的最佳執行計畫。首先我們將 DBLP 文件當中所有的關鍵字的數量透過建立索引時進行統計，並且將所有的關鍵字，依據其數量，分為高、中、低三個類別，然後選出該類別中的其中兩個關鍵字放入 Query 的資訊檢索限制中。表 5.4 為三個類別中，我們選出的關鍵字，以及該關鍵字的數量，在表 5.6 中，我們列出了 Q6、Q7、Q8 當中的標籤數量。

	Low Frequency (Q6)		Medium Frequency (Q7)		High Frequency (Q8)	
關鍵字	Air	Consider	Architecture	Knowledge	System	Us
數量	140	123	4564	4510	20798	15954

表 5.4：關鍵字在文件中出現的數量

	Q6、Q7、Q8	
標籤	booktitle	title
數量	364810	1590638

表 5.5：Q6 到 Q8 中的標籤在文件中出現的數量

在圖 5.5 中可以看出，隨著關鍵字頻率的增加系統的執行時間也會有所成長，主要是因為我們系統所選擇的執行計畫皆是採用關鍵字優先的方法。在這個實驗的狀況中，因為我們查詢數葉節點利用結構優先所截取的資料仍然大過於高頻率的關鍵字優先方法(參見表 5.4 和表 5.5)，所以最後仍都還是選用關鍵字優先的執行計畫。

	Best Plan	Selected Plan	Default Plan - KF	Default Plan - SF	Worst Plan	PSA
Q6 執行時間	0.06	0.06	0.07	2.17	3.08	0.03
Q7 執行時間	0.16	0.16	0.20	1.99	3.11	0.04
Q8 執行時間	0.41	0.41	0.43	2.917	3.08	0.03

表 5.6：關鍵字頻率之執行時間(Sec.)

	Best Plan	Selected Plan	Worst Plan
Q6	JOPLKKOPLKK	JOPLKKOPLKK	JOPTOPT
Q7	JOPLKKOPLKK	JOPLKKOPLKK	JOPTOPT
Q8	JOPLKKOPLKK	JOPLKKOPLKK	JOPTOPT

表 5.7：關鍵字頻率之最佳以及最差執行計畫

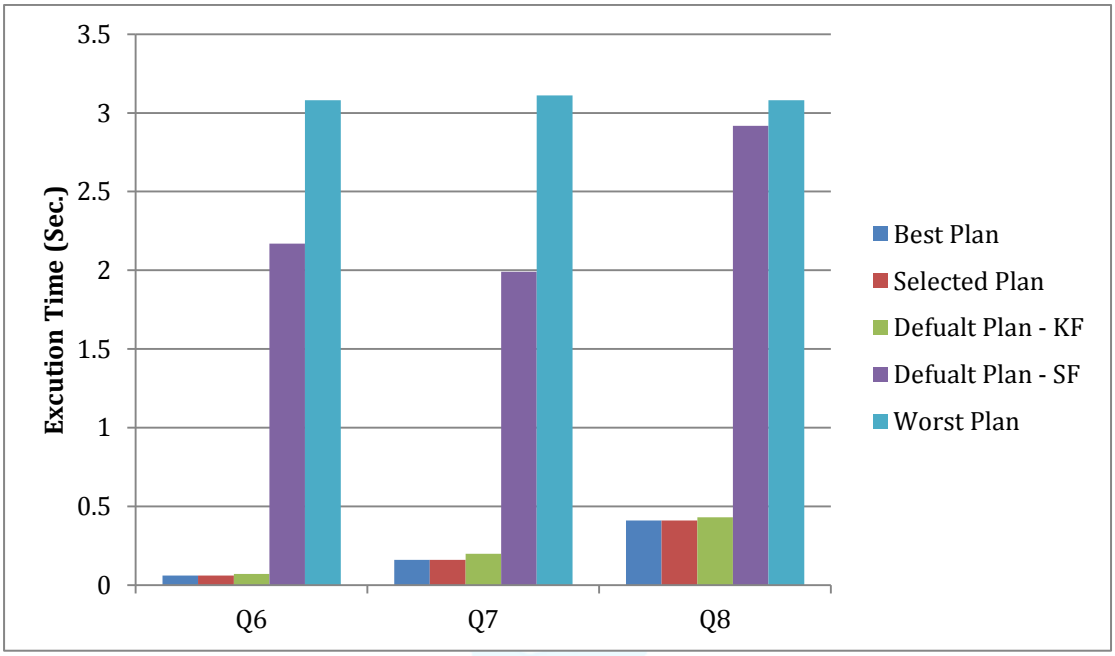

圖 5.6：關鍵字頻率之整體執行時間

5.3 標籤頻率

在 Q9、Q10、Q11 當中，我們選用了 DBLP 的文件來實驗標籤頻率的效能。同樣的，我們也將兩份文件中的標籤元素數量，透過統計資料，分為高、中、低三種類別，並且在每個類別中，選擇兩個標籤來進行查詢，標籤數量列表如表 5.8。

	Low Frequency (Q9)		Medium Frequency (Q10)		High Frequency (Q11)	
標籤	school	note	editor	publisher	title	author
數量	270	39	16038	6454	1590638	1296015

表 5.8：標籤在文件中出現的數量

	Best Plan	Selected Plan	Default Plan - KF	Default Plan - SF	Worst Plan	PSA
Q9 執行時間	0.04	0.04	0.12	0.04	0.18	0.03
Q10 執行時間	0.096	0.104	0.18	0.31	0.47	0.04
Q11 執行時間	0.28	0.28	0.28	2.97	4.74	0.03

表 5.9：標籤頻率之執行時間(Sec.)

	Best Plan	Selected Plan	Worst Plan
Q9	JPOTPOT	JPOTPOT	JOPLKKOPLKK
Q10	JPOTPOT	JOPLKKOPLKK	JPOTPOT
Q11	JPOLKKPOLKK	JOPLKKOPLKK	JPOTPOT

表 5.10：標籤頻率之最佳以及最差執行計畫

由實驗結果可以看出，當標籤頻率較低的時候，系統會選擇使用結構優先的方式執行，其中，又因為我們所下的資訊檢索限制主要都是在過濾標籤內是否存在特定關鍵字，所以先行判斷 ordered 限制式後，再去執行 P_match 時，P_match 所需要判斷的數量就會比直接先判斷 P_match 之後，再來判斷 ordered 限制來的少很多，因此系統針對查詢樹中的兩個葉節點選出的最佳執行計畫皆為先執行 O 之後再執行 P。

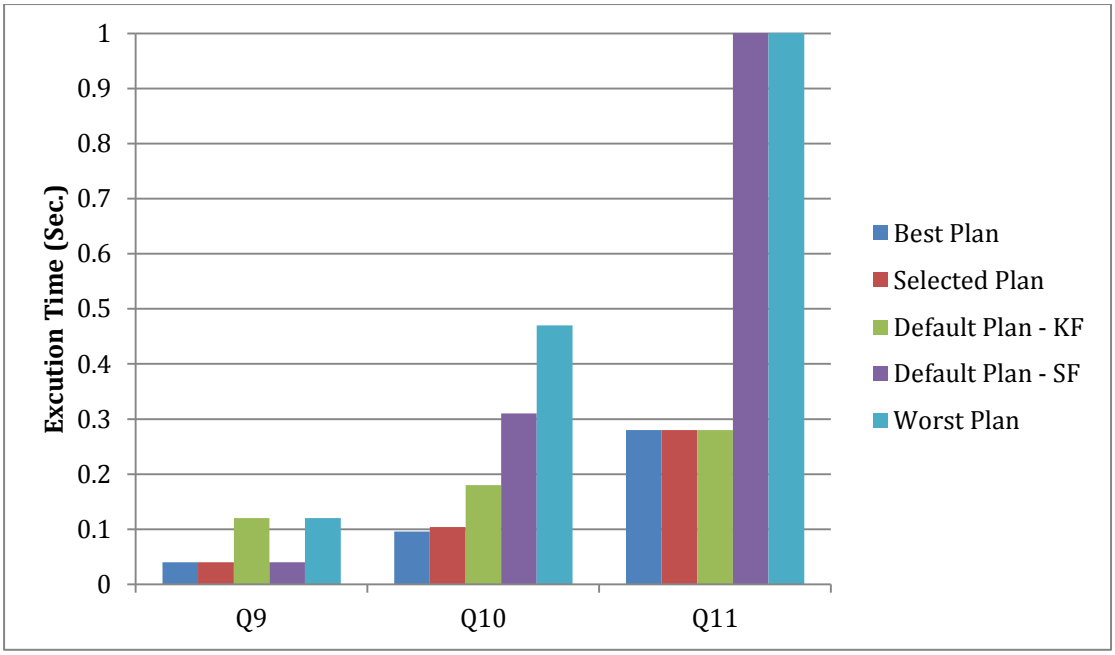

圖 5.7：標籤頻率之整體執行時間

5.4 不同高度節點限制

在 Q12、Q13、Q14 當中，我們給予一個固定的資訊檢索限制，但調整查詢樹的路徑，其中 Q14 的路徑最長，Q12 的路徑最短。圖 5.8 中可以看到，因為關鍵字優先所要擷取以及處理的資料量比結構優先來的少，如表 5.11 及表 5.12 所示，所以在這三組 Query 當中，系統仍然都是選擇關鍵字優先作為最佳執行計畫。另外，因為在 Q13 當中的標籤“Text”所包含的標籤元素資料較 Q14 的“description”以及 Q12 的“keyword”多，所以執行時間與其他兩個比較起來較長。至於關鍵字優先，則是因為我們所下的資訊檢索限制在三個 Query 當中都相同，因此執行時間並沒有太大的差異。

	Q12、Q13、Q14		
標籤	keyword	text	description
數量	362118	2921142	2339616

表 5.11：Q12 到 Q14 中的標籤在文件中出現的數量

	Q12、Q13、Q14		
關鍵字	master	attend	see
數量	2088	1211	14687

表 5.12：Q12 到 Q14 中的關鍵字在文件中出現的數量

	Best Plan	Selected Plan	Default Plan - KF	Default Plan - SF	Worst Plan	PSA
Q12 執行時間	0.45	0.45	0.45	0.6	1.32	0.04
Q13 執行時間	0.46	0.46	0.46	3.19	3.94	0.03
Q14 執行時間	0.46	0.46	0.46	2.43	2.63	0.03

表 5.13：不同高度節點限制之執行時間(Sec.)

	Best Plan	Selected Plan	Worst Plan
Q12	JDPLKKDPLKK	JDPLKKDPLKK	JDPTDPT
Q13	JDPLKKDPLKK	JDPLKKDPLKK	JDPTDPT
Q14	JDPLKKDPLKK	JDPLKKDPLKK	JDPTDPT

表 5.14：不同高度節點限制之最佳以及最差執行計畫

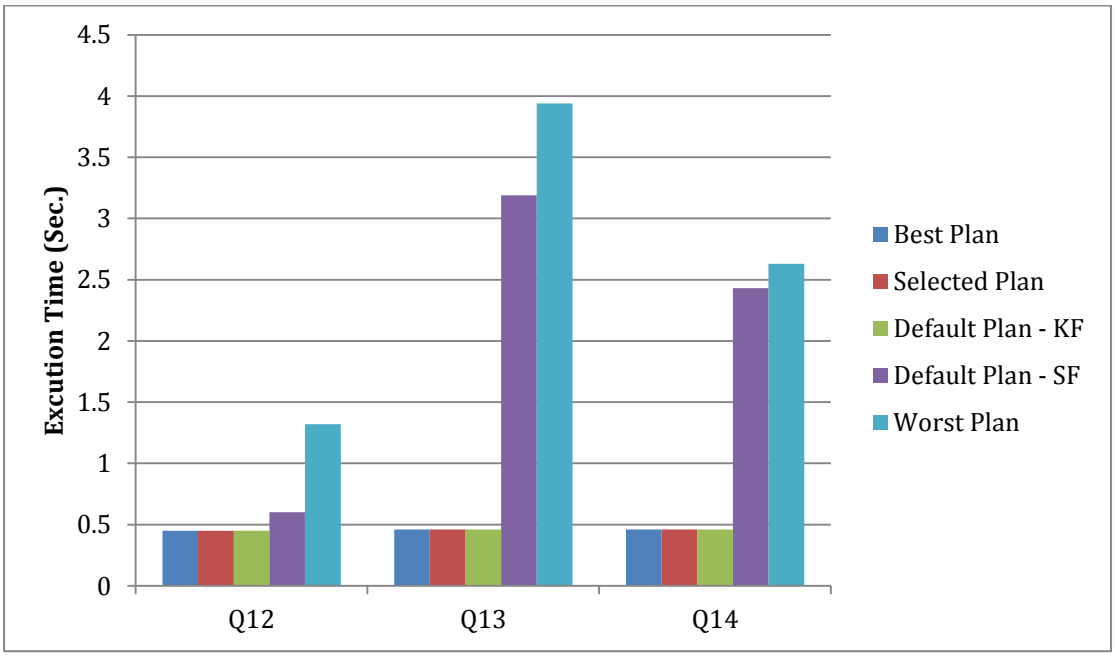

圖 5.8：不同高度節點限制之整體執行時間

5.5 不同節點距離限制

在這節的實驗中，我們固定查詢樹中的葉節點路徑，同時也固定資訊檢索限制中的關鍵字，唯一調整的，是資訊檢索限制中 distance 的距離。在 Q15、Q16、Q17 當中，我們分別將 distance 限制由>=5 調整成>=50000，以及>=50000000000。

	Best Plan	Selected Plan	Default Plan - KF	Default Plan - SF	Worst Plan	PSA
Q15 執行時間	0.36	0.36	0.37	3.28	4.08	0.03
Q16 執行時間	0.34	0.34	0.36	4.06	4.12	0.04
Q17 執行時間	0.36	0.36	0.37	4.05	4.14	0.03

表 5.15：不同節點距離限制之執行時間(Sec.)

	Q15、Q16、Q17	
標籤	text	shipping
數量	2921142	151694

表 5.16：Q15 到 Q17 中的標籤在文件中出現的數量

	Q15、Q16、Q17		
關鍵字	master	attend	see
數量	2088	1211	14687

表 5.17：Q15 到 Q17 中的關鍵字在文件中出現的數量

	Best Plan	Selected Plan	Worst Plan
Q15	JDPLKKOPLKK	JDPLKKOPLKK	JDPTOPT
Q16	JDPLKKOPLKK	JDPLKKOPLKK	JDPTOPT
Q17	JDPLKKOPLKK	JDPLKKOPLKK	JDPTOPT

表 5.18：不同節點距離限制之最佳以及最差執行計畫

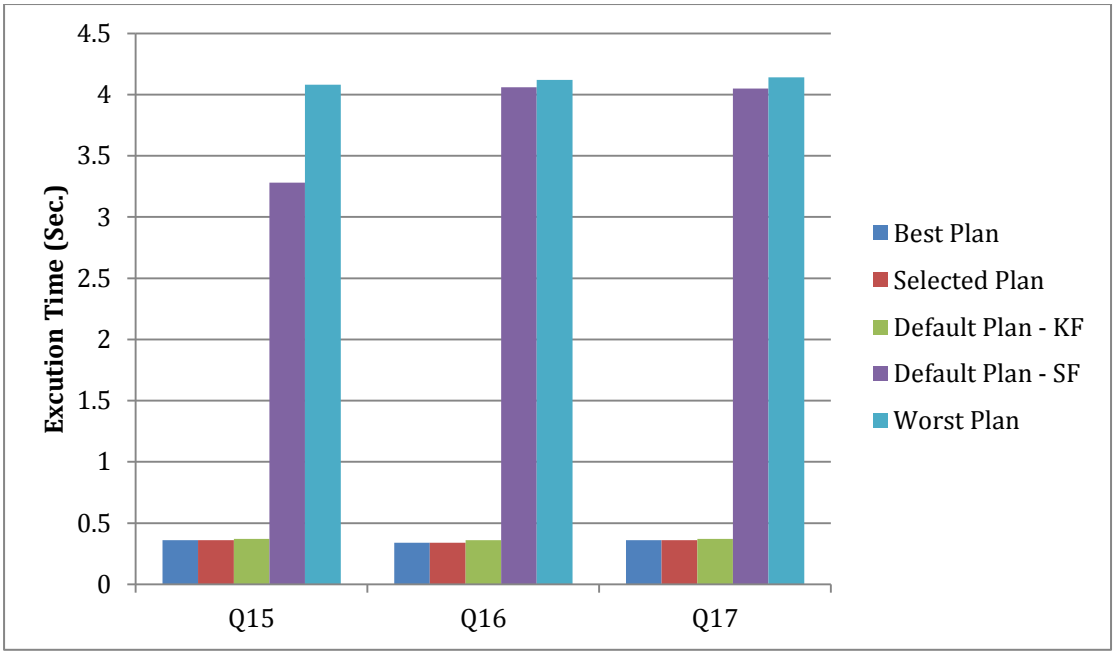

圖 5.9：不同節點距離限制之整體執行時間

由實驗數據可以看出，當 distance 的距離限制越遠時，系統選出的最佳執行計畫執行的時間便會略微增加，主要是因為系統選擇關鍵字優先的方法執行，而在關鍵字優先的方法中，當距離增加時，產生的 LCA 也就會越高，因此時間也就會增加。在結構優先以及關鍵字優先之間，因為結構優先在 T_match 取得的資料仍然是比關鍵字優先利用 K_match 取得的資料數量來的多，如表 5.16 和表 5.17 所示，因此 Cost Model 的計算結果還是比關鍵字優先來的大。

5.6 遞增的路徑限制

在這節的實驗中，我們將查詢樹當中的路徑限制數量增加，而結果也顯示，當路徑限制增加時，因為需要處理的結構變得複雜，同時對於關鍵字優先的方法則是增加了判斷資訊檢索限制的數量，所以兩者的執行時間都有增加，但是因為關鍵字優先所需擷取的資料量，還是比每增加一個 branch node 就得多擷取一整個標籤資料的結構優先來的少，如表 5.20 和表 5.21 所示，所以系統選擇的最佳執行計畫還是關鍵字優先。而在關鍵字優先的執行計畫中，又因為我們所下的資訊檢索限制皆為兩個相同關鍵字的 ordered 限制，事實上即代表截取出來的 SCU Table 已經符合資訊檢索的限制，所以在執行計畫上，選擇先判斷路徑，再判斷 ordered 限制的計畫效率最高。

	Best Plan	Selected Plan	Default Plan - KF	Default Plan - SF	Worst Plan	PSA
Q18 執行時間	0.45	0.45	0.46	3.61	5.62	0.03
Q19 執行時間	0.47	0.47	0.47	3.65	5.68	0.03
Q20 執行時間	0.49	0.49	0.54	3.73	5.92	0.04

表 5.19：遞增的路徑限制之執行時間(Sec.)

	Q18、Q19、Q20		Q19	Q20
標籤	text	shipping	location	from
數量	2921142	151694	40041	62838

表 5.20：Q15 到 Q17 中的標籤在文件中出現的數量

	Q18、Q19、Q20			Q19	Q20
關鍵字	master	bound	description	Netherland	June
數量	2088	532	11193	65	15

表 5.21：關鍵字在文件中出現的數量

	Best Plan	Selected Plan	Worst Plan
Q18	JOPLKKOPLKKOPLK K	JOPLKKOPLKKOPLK K	JOPTOPTOPT
Q19	JOPLKKOPLKKOPLK KOPLKK	JOPLKKOPLKKOPLK KOPLKK	JOPTOPTOPTOPT
Q20	JOPLKKOPLKKOPLK KOPLKKOPLKK	JOPLKKOPLKKOPLK KOPLKKOPLKK	JOPTOPTOPTOPTOPT

表 5.22：遞增的路徑限制之最佳以及最差執行計畫

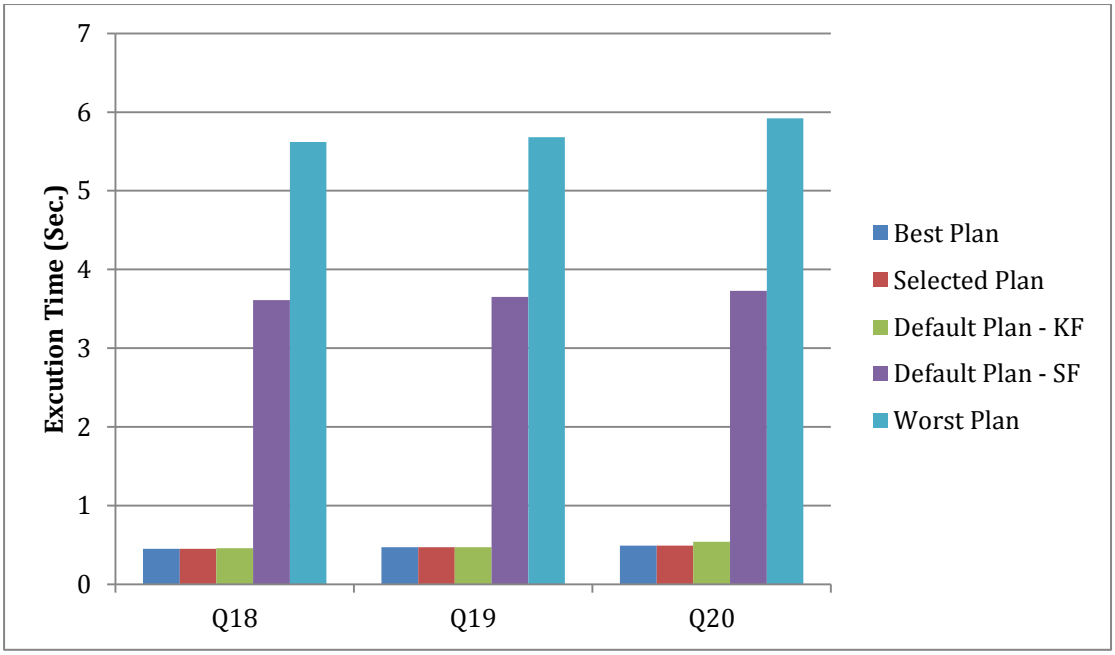

圖 5.10：遞增的路徑限制之整體執行時間

5.8 Heuristic Rules

透過前面一系列的實驗，我們可以發現雖然在大多數的狀況下，選擇關鍵字優先的方法都可以產生比較快速的結果，但是當資訊檢索限制中所需要截取的關鍵字數量，比使用結構優先方法時需要截取的標籤元素數量多時，因為兩者在取得資料的係數幾乎相同，並且皆比其他部份的係數來的大，所以這時候兩者截取的資料量，變成為影響分數的主要因素。由此，我們定義第一個 heuristic rule：

若關鍵字數量小於標籤數量，則選擇 KF 的 default plan。(Rule 1)

另外在遞增 IR 限制的實驗中，我們發現，因為在增加資訊檢索限制時，對於關鍵字優先來說，會需要增加計算 LCA 的時間，在資訊檢索限制未達一定數量前，計算 LCA 所增加的 cost，並不會影響關鍵字優先的整體分數，但是因為計算 LCA 的次數與資訊檢索限制的關係為 2N-1，其中 N 為資訊檢索限制的數量，另外，圖 4.18 中，我們可看出 LCA 的係數比 I/O 的係數小一個 order，，因此我們也可以針對這樣的狀況給予一個 Heuristic Rule 如下：

若 N>20，則選擇 SF 的 Default Plan。 (Rule 2)

綜合 rule 1 和 rule 2，我們設計以下的 heuristic plan：

If(sum(K_match) < sum(T_match)) or N <2 0

Choose default plan – KF

Else

Choose default plan – SF

我們利用前面 5.1 實驗中的 Query 重新利用 heuristic rule 執行一次，得到以下的結果：

	Best Plan	PSA Selected Plan	Heuristic Plan
Q1_H 執行時間	0.77 (KF)	0.77 (KF)	0.77 (KF)
Q2_H 執行時間	0.99 (KF)	0.99 (KF)	0.99 (KF)
Q3_H 執行時間	1.33 (KF)	1.34 (KF)	1.378 (KF)
Q4_H 執行時間	1.37 (KF)	1.37 (KF)	1.952 (KF)
Q5_H 執行時間	1.929 (SF)	1.929 (SF)	1.929 (SF)
Q6_H 執行時間	0.06 (KF)	0.06 (KF)	0.06 (KF)
Q7_H 執行時間	0.16 (KF)	0.16 (KF)	0.16 (KF)
Q8_H 執行時間	0.41 (KF)	0.41 (KF)	0.41 (KF)
Q9_H 執行時間	0.04 (SF)	0.04 (SF)	0.04 (SF)
Q10_H 執行時間	0.096 (SF)	0.104 (KF)	0.18 (KF)
Q11_H 執行時間	0.28 (KF)	0.28 (KF)	0.28 (KF)
Q12_H 執行時間	0.45 (KF)	0.45 (KF)	0.45 (KF)
Q13_H 執行時間	0.46 (KF)	0.46 (KF)	0.46 (KF)
Q14_H 執行時間	0.46 (KF)	0.46 (KF)	0.46 (KF)
Q15_H 執行時間	0.35 (KF)	0.35 (KF)	0.36 (KF)
Q16_H 執行時間	0.34 (KF)	0.34 (KF)	0.34 (KF)
Q17_H 執行時間	0.34 (KF)	0.34 (KF)	0.34 (KF)
Q18_H 執行時間	0.45 (KF)	0.45 (KF)	0.45 (KF)
Q19_H 執行時間	0.47 (KF)	0.47 (KF)	0.47 (KF)
Q20_H 執行時間	0.49 (KF)	0.49 (KF)	0.49 (KF)

表 5.23：使用 Heuristic Rule 之執行時間(Sec.)

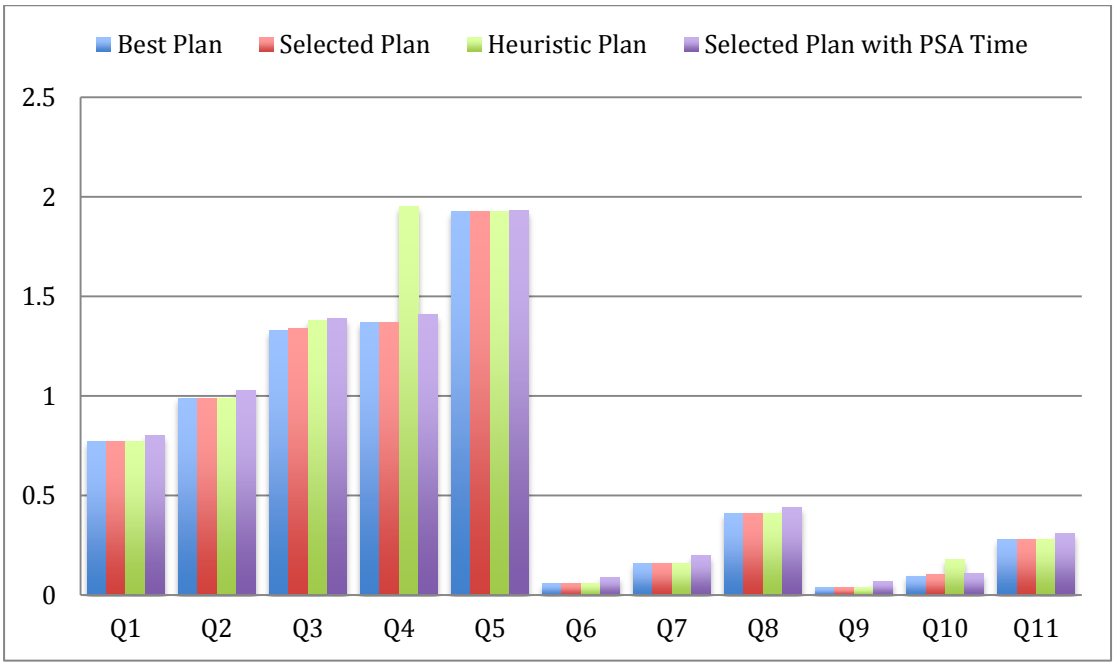

圖 5.11：DBLP 107MB Dataset 使用 heuristic rule 實驗數據

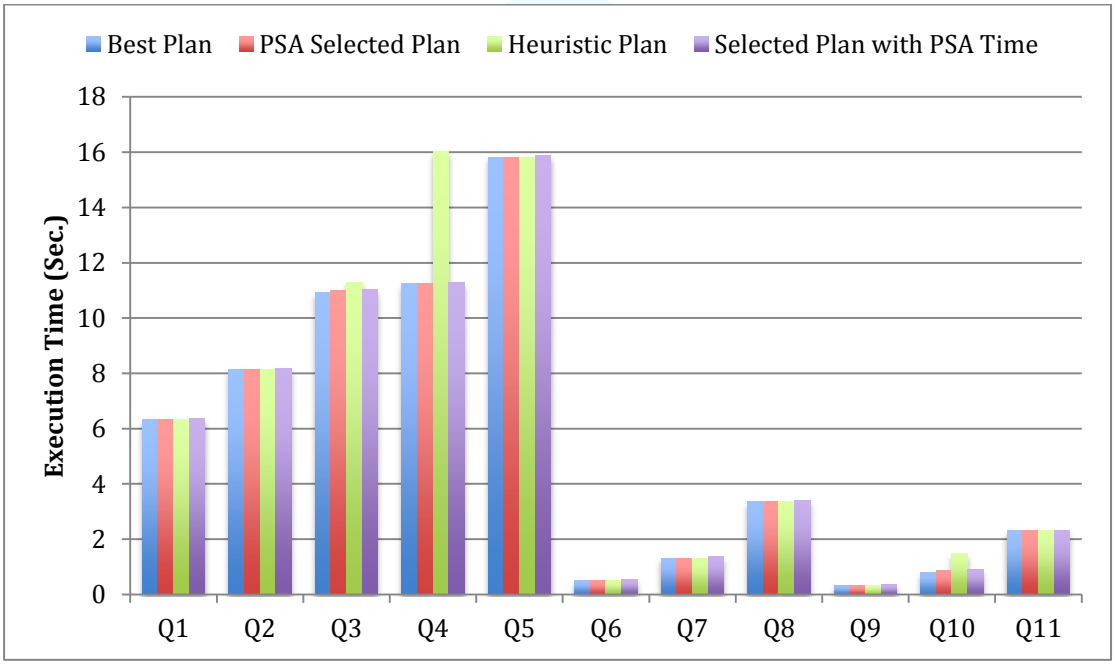

圖 5.12：DBLP 872MB Dataset 使用 heuristic rule 實驗數據

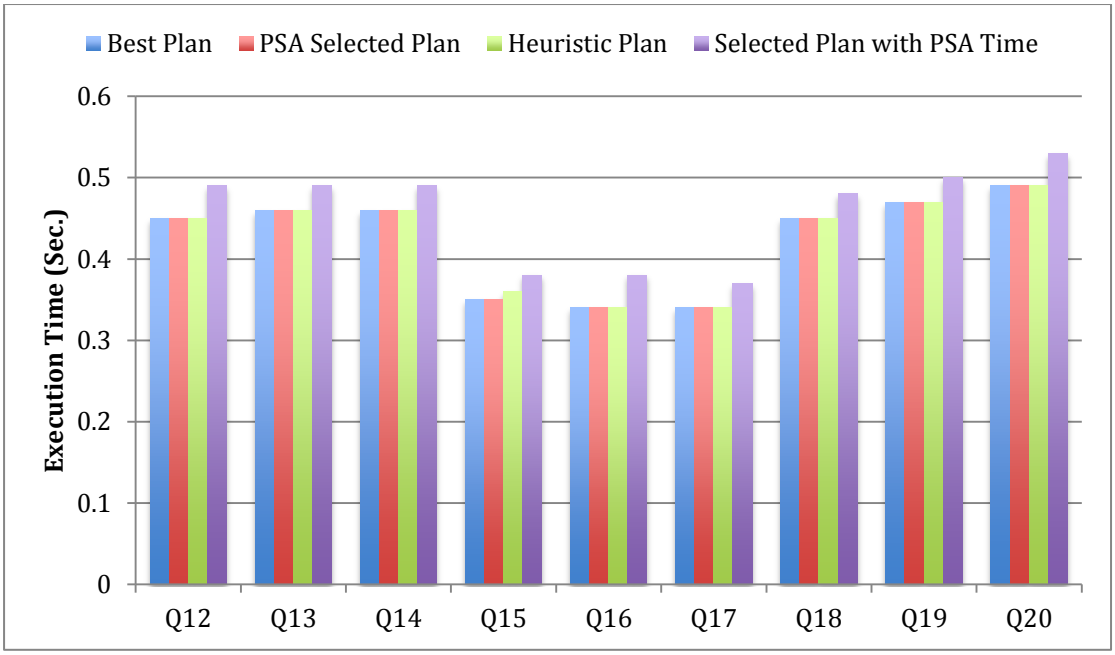

圖 5.13：XMark Dataset 使用 heuristic rule 實驗數據

由實驗結果可以發現，在 DBLP 當中，heuristic plan 有 72%為最佳 plan，其中的 Q3_H、Q4_H、Q10_H 所選擇的 heuristic plan 不正確外，其餘的 heuristic plan 皆能正確選出實際最佳的執行計畫。在 Q3_H 中 heuristic plan 所選出的執行計畫所花的時間比 PSA 選出的時間多出 0.038 秒，在 Q4_H 中 heuristic plan 則是比 PSA 多出 0.612 秒，而 Q10_H 中，Heuristic Plan 所選出的執行計畫所花的時間為 0.18 秒，比 PSA 所選出的 0.104 秒多 0.076 秒。另外在 XMark 當中，則是有 88.8%為最佳的 plan，其中 Q15_H 的 heuristic plan 則是比 PSA 選出的多出 0.01 秒。

使用 PSA 所選出的最佳執行計畫中，DBLP 的部分有 90%可以選到實際的最佳執行計畫，而在使用 heuristic plan 時，則只有 72%可以選到實際最佳的執行計畫，另外在 XMark 上，使用 PSA 有 100%準確率，但是 heuristic plan 則只有 88.8%，所以雖然 PSA 會比 heuristic plan 多花一點時間，但是準確率還是比 heuristic plan 高得多。然而在考慮 PSA 執行時間後，Heuristic Plan 執行的時間比考慮 PSA 之後的時間還要多的，在 DBLP 當中所佔的比例為 27%，在 XMark 則完全沒有。所以 heuristic plan 在絕大部分是比較有效率的。

第六章 結論與未來方向

在本論文中，針對結構優先，以及關鍵字優先查詢處理建立了數個具有獨立性的運算子，使得這兩種作法中的幾個重要步驟，可以任意的調換順序。有了這些運算子後，本論文進而提出了適用於這兩種方法的 rewriting rule。另外，我們也提出了一套可以針對這兩個方法的 cost model，並且透過實驗證明，在大部分的情況下，我們的 cost model 都能找出正確的找出最佳的執行計畫。同時在實驗的過程中，我們也找出了適用於這兩個方法的 heuristic rule，可以減少 PSA 所花的時間。

本論文的作法，可以針對兩種不同的做法，提供了統一的查詢，無須針對不同的做法，改變資料儲存的結構。而且由實驗的結果可以看出，透過我們的系統可以充分發揮結構優先以及關鍵字優先這兩種作法各自的長處，使得整體的效率提昇。

本論文未研究的方向，可以在針對 XML 文件進行前處理時，增加不同性質的統計數據，以便增加 cost model 的準確度。

參考文獻

[AYJ03] Shurug Al-Khalifa, Cong Yu, H. V. Jagadish, "Querying Structured Text in an XML Database", In Proceedings of the SIGMOD Conference, Jun. 2003.

[ACD06] 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.

[BCFH05] Peter Buneman, Byron Choi, Wenfei Fan, Robert Hutchison, Robert Mann, Stratis Viglas, "Vectorizing and Querying Large XML Repositories", In Proceedings of the ICDE Conference, 2005.

[CKS06] SungRan Cho, Nick Koudas, Divesh Srivastava, "Meta-data Indexing for XPath Location Steps", In Proceedings of the SIGMOD Conference, June 27–29, Chicago, Illinois, USA, 2006.

[GSBS03] 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.

[GORS08] Giorgio Ghelli, Nicola Onose, Kristoffer Rose, Jérôme Siméon, “XML Query Optimization in the Presence of Side Effects”, In Proceeding of SIGMOD Conference, 2008.

[GCV09] Haris Georgiadis, Minas Charalambides, Vasilis Vassalos, “Cost Based Plan Selection for XPath”, In Proceeding of SIGMOD Conference, 2009.

[KKNR04] 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.

[KBMK09] Riham Abdel Kader, Peter Boncz, Stefan Manegold, Maurice van Keulen, “ROX：Run-time Optimization of XQueries”, In Proceeding of SIGMOD Conference, 2009.

[LLCC05] 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.

[MW99] Jason McHugh, Jennifer Widom, "Query Optimization for XML", In Proceedings of VLDB Conference, 1999.

[MN10] Sebastian Maneth, Kim Nguyen, "XPath Whole Query Optimization", In Proceedings of VLDB Conference, 2010.

[WPJ03] Yuqing Wu, Jignesh M. Patel, H. V. Jagadish , "Structural Join Order Selection for XML Query Optimization", In Proceedings of the ICDE Conference, 2003.

[XP05] Yu Xu, Yannis Papakonstantinou, "Efficient Keyword Search for Smallest LCAs in XML Databases", In Proceedings of the SIGMOD Conference, 2005.

[XP05] Yu Xu, Yannis Papakonstantinou, "Efficient Keyword Search for Smallest LCAs in XML Databases", In Proceedings of the SIGMOD Conference, 2005.

[ZC03] Zhimin Chen , "From Tree Patterns to Generalized Tree Patterns: On Efficient Evaluation of XQuery", In Proceedings of the VLDB Conference, 2003.

[吳 09] 吳政儀, “支援具有複雜關鍵字限制之 XML 查詢系統”, 碩士論文, 國立臺灣海洋大學資訊工程系, 2009