# Semantic Indexing for Chinese Structured Documents

Student: Chih-Hsuan Tseng Advisor: Dr. Hao-Ren Ke, Dr. Wei-Pang Yang.

Institute of Computer and Information Science

National Chiao Tung University

## ABSTRACT

In this thesis, we propose a information retrieval scheme that combines indexing for structured documents and semantic indexing. Indexing for structured documents can index the documents with embedded document structures. By using characteristics of K-ary trees, we can store the element data and provide fast element access. On the other hand, semantic indexing constructs a conceptual space or knowledge space by using semantic matrices. Through the idea of conceptual space and semantic network, we expect that traditional information retrieval will be evolved into knowledge retrieval. In this thesis, we construct several evaluations to assess both traditional indexing scheme and our integrated indexing scheme. Although the integrated indexing scheme takes more time to build indexes, it can present more information relevant to user interests.

# 中文結構化文件之語意索引

研究生: 曾志軒　　指導教授: 柯皓仁博士，楊維邦博士

國立交通大學資訊科學研究所

## 摘要

在本篇論文中，我們提出一套結合結構化文件索引(Indexing for Structured Documents)及語意索引(Semantic Indexing)的檢索系統。結構化文件索引是針對文件內包含有結構化資訊的文件做索引，利用 K-ary 樹狀結構的特性來儲存元素資料並提供快速的檢索機制。語意索引則是利用語意矩陣將文件中關鍵詞鍵建置為一個「概念空間(Concept Space)」或「知識空間(Knowledge Space)」。關鍵詞鍵的概念空間乃是知識的一種表現形式，且能以語意網路的方式來描述。透過概念空間以及語意網路，我們期能將傳統的資訊擷取提升至知識擷取的層次。在論文中，我們分別針對結構化文件索引方法與結合結構化文件索引及語意索引的方法做關鍵詞檢索。效益評估結果顯示，結合兩種索引之方法雖然較傳統結構化文件需要較多的檢索處理時間，但卻能找出更多符合使用者興趣的資訊。

# 誌謝

感謝兩年來指導教授柯皓仁老師及楊維邦老師在各方面的細心指導與照顧，使我不只在課業與研究上獲得許多寶貴的指點，在生活與做人處事上，更給了我不少的影響與啟示。

此外還要特別感謝實驗室學長們對我研究上的啟發與指導，提供我許多寶貴的建議，使得我論文得以順利的完成。還有實驗室的夥伴們，對我這兩年來的關懷與照顧。

最後，還要感謝我親愛的家人與朋友們長久以來的支持與鼓勵，使我能專心致力於研究，並得以順利完成學業。

# 目錄

# 圖目錄

# 表目錄

# 方程式目錄

# 第一章　簡介

## 第一節　資訊擷取與索引

隨著電腦科技與數位化技術的迅速發展，網際網路已經成為全球新一代的主角，大量的數位化文件透過網際網路快速而廣泛地傳遞，使得網際網路無形中成為一個儲存各種資源的龐大文件庫及資料庫；在此資訊爆炸的網路時代裡，人們面對的不再是訊息的匱乏，而是過度發展之後，資訊過於龐大、複雜的問題。在這種情形下，為了讓使用者能夠在龐大的資料中找到感興趣的資訊，資訊擷取的相關研究因而蓬勃發展。

自動化資訊擷取系統的發展主要是為了幫助檢索龐大的資料量，資料可能是純文字、影片或是音樂。一般而言，傳統的索引只能提供一些事先建置好的索引詞彙供檢索且必須經由學科專家來編排索引；而資訊擷取的目的則希望能透過自動化的處理過程，幫助人們定位出有用的資訊，提供使用者感興趣的資料。然而，資訊並不是知識，如何將網際網路上龐大的資訊經過綜合、整理及分析之後，推演出有價值的知識，乃是在此知識經濟時代非常重要的研究課題。

有鑑於此，本論文擬運用資訊科技來發展知識擷取相關技術與系統。本論文所討論的重點著重在利用可擴展標示語言(XML)製作之中文文件，利用資訊擷取的技術，抽取出文件的關鍵詞做為索引，並提供給使用者檢索之用。檢索機制會分析使用者所輸入的關鍵詞，與已建置好的索引作相似性比對，最後按照相似程度加以排列並選出符合使用者興趣的文件資訊。在檢索運作的流程中，索引扮演著引導者的角色，它可以提供系統快速找到每一個關鍵語的能力，而不需與文件本身的文字內容作逐字比對，讓系統可以對使用者的要求作最快速的回應。

由於索引的重要性及必要性，其相關研究已成為近年來資訊擷取處理中極重要的課題。在傳統的索引技術中，通常都是針對文件本身的文字內容做前置處理

以建立索引。文件中所有的文字內容先經過斷詞切字，找出可能代表該文件的詞鍵(Term)，並計算各個詞鍵的出現頻率以找出真正能代表該文件的詞鍵；接著再經過索引演算法將這些詞鍵建置成索引；最後，這些處理後的索引詞鍵(Index Term)即代表著該文件的內容，並可提供給使用者作檢索之用。

上述一連串的步驟主要在說明傳統資訊擷取中處理純文字文件的流程。隨著資訊技術的發展，傳統的純文字文件格式已經無法達到大量管理以及資料交換的要求，故 World Wide Web Consortium(W3C)在 1998 年提出了可擴展標記語言(eXtensible Markup Language, XML)以便於網際網路上文件的交換與管理。XML 文件主要的特色就在於包含了文件中的結構化資訊，一個 XML 文件可以轉換成一個文件樹狀結構(Document Tree Structure)，在文件樹狀結構中的每一個節點代表著一個 XML 標籤，而標籤(Tag)與標籤之間存在階層式的關係。結構化的資訊提供了比傳統的文件內容更詳細的資料，善加利用結構化資訊將使得檢索機制更為強大，故將傳統的索引技術應用在結構化文件中已在近年來吸引相當多的研究工作 [Wolff00, Chow99, Han99, Kasukawa99, Dao98, Myaeng98, Shin98, Poullet97, Lee96, Wilkinson94]。

在語意索引(Semantic Indexing)的建立方面，由於傳統的索引技術僅提供文件內的單一關鍵字詞供檢索，無法讓使用者找到與關鍵字詞語意上相類似的文件，因此有了語意索引的方法產生。語意索引除了能找出文件內的關鍵字詞外，並能夠找出所有語意概念上相近的關鍵字詞，這使得索引的技術更為強大，也提供給使用者更方便的檢索機制。在[Chung99]這篇論文中，將語意索引應用在一個龐大的文件資料庫中以驗證其可行性。

## 第二節　研究動機

結構化文件索引的建立，會先分析該文件的階層式架構，並將階層式架構中每一節點上的內容存入索引之內以供檢索。基於這樣的設計，使用者得以針對結構化文件中的節點資訊作檢索，如使用者欲檢索的資訊為「我想找『摘要』中含

有『數位圖書館』的文件。」則系統會根據結構化文件索引找到所有摘要中包含有數位圖書館的文件，並回傳給使用者以供瀏覽。這種檢索方式的好處就是可以針對某些節點做檢索，而不僅僅針對整個文件做檢索，不但快速且更能提供較深層的檢索。

再者，由於目前結構化文件索引的建立僅是針對文字內容作處理，所能提供檢索的僅僅是按關鍵詞檢索，若將語意索引的建置方法融入其中，不僅能提供關鍵詞檢索，更可提供語意檢索的能力。比如說當使用者欲檢索「數位圖書館」，其目的除了找出含有「數位圖書館」的文件，也可能希望能找到語意相近的資訊，如「電子圖書館」、「內容管理」、「知識管理」、「網際網路」或「資料擷取」等。

綜合上述的概念，本篇論文中擬加入文件的階層式結構資訊在索引裡面，並在建立索引的過程中應用語意索引的方法，建立起文件中關鍵詞鍵之概念空間，所謂關鍵詞鍵的概念空間乃是知識的一種表現形式，且能以語意網路的方式來描述。透過概念空間以及語意網路，我們期能將傳統的資訊擷取提升至知識擷取的層次。

## 第三節　研究目的

本論文主要目的是針對結構化文件提出一套語意索引的演算法，此演算法將 XML 文件結構化資訊與傳統索引技術加以整合，在不影響傳統檢索的效率下，將文件的結構化資訊納入索引的建置中；並融合語意式索引的方法，分析所有索引過程中產生的詞鍵，建構出兩兩詞鍵的概念空間，以提供知識擷取的檢索能力，期許能將傳統的資訊擷取提升到知識擷取的層次。

## 第四節　論文架構

本論文第二章介紹相關研究工作，第三章描述如何結合結構化文件索引與語意索引，第四章說明系統實作與結果分析，藉以驗證本論文的可行性，第五章是結論與未來研究的方向。

# 第二章　相關研究工作

本章說明與本論文相關的研究工作，主要可分為兩大類，一類為結構化文件索引的建立，例如[Lee96]、[Shin98]與[Han99]等論文。這一類的論文著重於如何針對結構化文件建置索引，或設計符合結構化文件特性的資料結構；另一類則是語意索引的建立，討論如何利用語意矩陣將兩兩詞鍵建構成一個概念空間以供檢索，如[Chung99]與[Bourret00]等論文。圖 1 中分別列出這些相關研究工作。

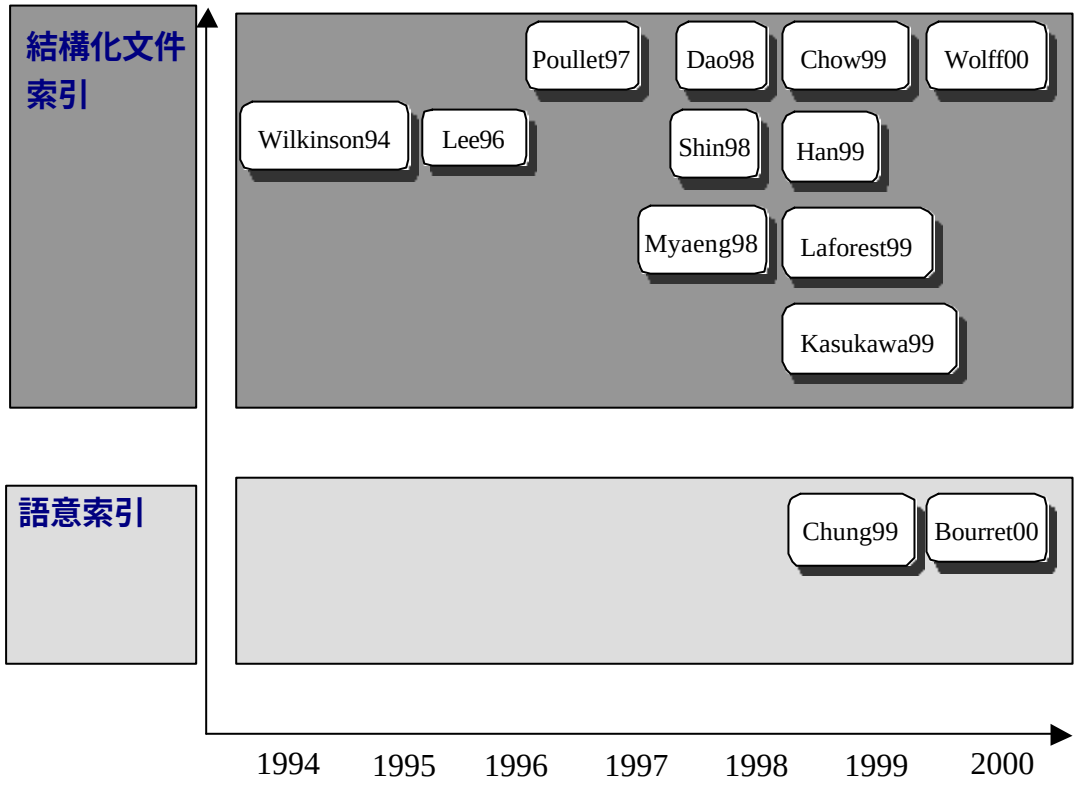


圖 1：相關研究工作

## 第一節　結構化文件索引的建立 (Indexing for Structured Documents)

對文件的檢索可分為針對文件內容的檢索(Content Query)與針對結構檢索(Structure Query)。Content Query 只能檢索文件包含的文字內容，無法檢索任何與文件結構有關的資訊。而 Structure Query 則可利用文件本身的結構資訊，提供使用者較深層的檢索，例如說可以檢索文件中第二段的內容是否含有「知識擷取」

的文字、第三章是否含有「數位圖書館」等等。這種可以檢索至文件結構資訊的檢索就稱為結構檢索。

要達到結構檢索的要求，首先必須針對結構化文件的特性作索引。在結構化文件索引的建立中，主要是提出新的索引結構以適用於結構化文件。在索引結構的設計方面，基本上是以[Lee96]所提出的 K-ary Tree 為典範，K-ary Tree 乃是一個完整的樹狀結構(Complete Tree Structure)，而 K 所代表的即為在文件樹狀結構中的最大分支數。[Lee96]賦予每個節點一個「單一元素識別號(Unique Element Identifier, UID)」，代表該節點在階層式架構中的絕對位址，透過 UID 只需用簡單的運算便可快速地找到該節點的父節點或子節點並擷取所需資料。繼[Lee96]後發表的[Shin98]則是將 UID 的概念擴充為整體元素識別號(General Element Identifier, GID)以適用於多份文件中，並提出一個由下而上的檢索機制(Button Up Scheme, BUS)來檢索建置好索引後的文件。以下茲針對[Lee96]及[Shin98]作進一步說明。

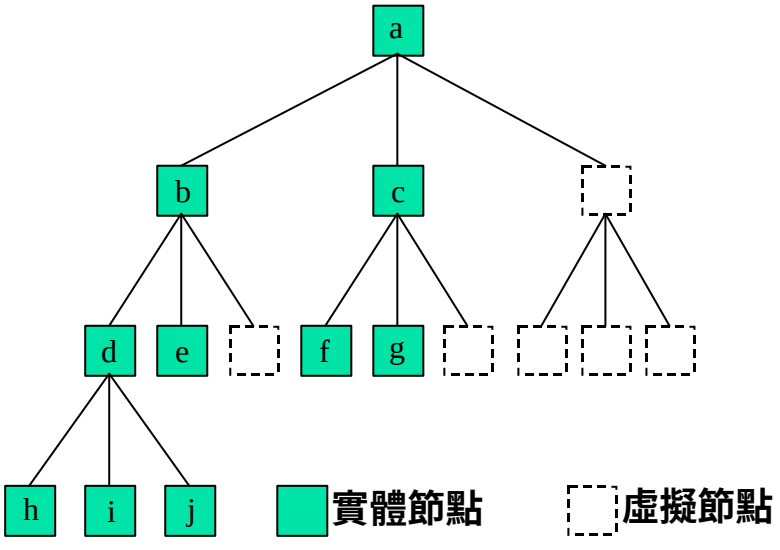


圖 2：3-ary tree 與結構化文件的對應圖

為了提供結構檢索，[Lee96]這篇論文中，首先設計了 UID 來代表各個節點在文件樹狀結構中的位置。由於每一個結構化文件都能夠展開成為一個階層式的樹狀結構，故作者將一份結構化文件對應到一個完整的樹狀架構(Complete K-ary

Tree)上，並以該文件最大的分支數作為此樹狀結構的分支數。由此樹狀結構對應到原始結構化文件來看，若節點存在於原始結構化文件則稱為實體節點(Real Node)，反之則稱為虛擬節點(Virtual Node)。將結構化文件對應到完整的樹狀結構後，再根據此樹狀結構上的節點依序將 UID 配置給每一個實體節點。如 圖 2 所示為一個 3-ary Tree，每一個內部節點都有三個分支，由實線繪成的方框即代表實體節點，但為了配置 UID，將虛擬節點也加入 3-ary Tree 之中，最後根據節點的位置配置每一個實體節點單一識別號如表格 1 所示。

| Element | UID | Element | UID |
|---|---|---|---|
| a | 1 | f | 8 |
| b | 2 | g | 9 |
| c | 3 | h | 14 |
| d | 5 | i | 15 |
| e | 6 | j | 16 |

表格 1：UID 配置表

利用 UID 來包含結構化文件的結構資訊有何好處呢？由於 K-ary Tree 為完整樹狀結構，故利用簡單的數學運算便能快速地計算出父節點與子節點的 UID，進而得以快速存取節點，這便是[Lee96]中利用 K-ary Tree 來儲存結構化文件資訊的主要目的。方程式 1 為父節點 UID 的計算公式，方程式 2 為子節點 UID 的計算公式。在方程式 1 與中，$i$ 所代表的是本身節點的 UID，$k$ 代表的則是此 K-ary樹狀結構的分支數。同樣地在方程式 2 中，$i$與$k$的意義不變，而 $j(1 \le j \le k)$ 則代表該節點為其父節點的第幾個分支。

$$parent(i) = \left\lfloor \frac{i-2}{k} + 1 \right\rfloor$$

方程式 1：計算父節點 UID 之公式

$$child(i, j) = k(i-1) + j + 1$$

方程式 2：計算子節點 UID 之公式

以表格 1 的範例來看，節點 d 為節點 h 之父節點，又節點 h 的 UID 為 14，代入方程式 1 可知其父節點 UID 應為 $parent(14)=\left\lfloor \frac{14-2}{3}+1 \right\rfloor =5$，的確符合節點 d 之 UID；反之由於節點 h 為節點 d 之第一個子節點，代入方程式 2 可知其 UID 為 $child(5,1)=3(5-1)+1+1=14$，同樣亦可驗證了此公式的正確性。

除了提出 UID 的概念來快速存取之外，[Lee96]並提出了下列五種不同的索引結構：

1. ANWR (All Nodes With Replication)：詳細記錄包含每一個關鍵字詞出的所有節點，相當浪費儲存空間。
2. ALWR (All Levels With Replication)：和 ANWR 相同，也記錄包含每一個關鍵字詞的所有節點，但在樹狀結構中會根據不同的深度將其切成不同的群組，與 ANWR 類似均會重複記錄節點資訊。
3. LNON (Leaf Nodes Only)：只記錄包含關鍵字詞的最底層節點。
4. ANOR (All Nodes Without Replication)：由 LNON 進化而來的索引結構，若發現關鍵字詞包含在同一層的所有節點，則將此關鍵字詞提升(Promote)到父節點上，如果提升的關鍵字詞夠多的話，將會省下可觀的儲存空間，也會提供更有效率的檢索，故此索引結構被該論文評定為最佳的索引結構。
5. RNON (Root Node Only)：僅儲存關鍵字詞出現的父節點，此方法雖然非常節省空間，但無法表現出結構化文件應有的結構特性，故不是一個好的索引架構。

舉例說明，若原本的結構化文件結構如圖 3 中的樹狀結構所示，則節點 A 儲存的索引資訊應為{personal, female, girl, woman}。Index(X)則為節點 X 中所存的索引，每一個節點都有必須儲存的資訊，但有些資訊會重複出現在不同的節點中，若我們直接將其記錄下來，勢必會浪費大量的儲存空間，故[Lee96]利用

ANOR 的架構(見圖 3 (b))以刪除重複儲存的資訊，大量地減少儲存所需空間。在經過詳細的評估與分析後，也證實了該方法的效率與可行性。

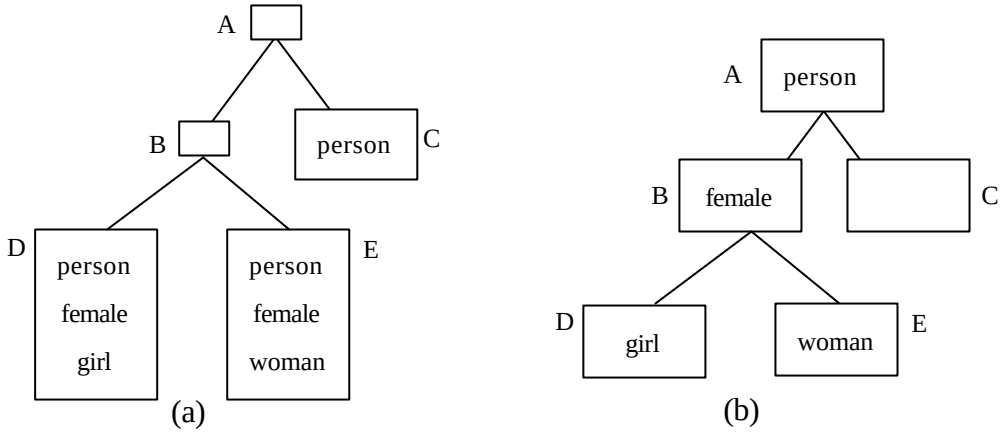


圖 3：結構化文件索引範例

由於 UID 的設計主要是針對單一文件的索引，無法處理大量文件，故繼[Lee96]發表後，[Shin98]提出了由下至上的的檢索機制(Bottom Up Scheme, BUS)以處理大量結構化文件，其中的關鍵技術便是在 UID 的設計中多了一個文件識別號(Document Identifier, DID)，將 UID 擴展成為 GID (General Element Identifier)，加入了 DID 之後，就能夠快速地找到關鍵字詞所在的文件，再配合原有的 UID 更可找到直接抓到文件內關鍵字詞的位置。

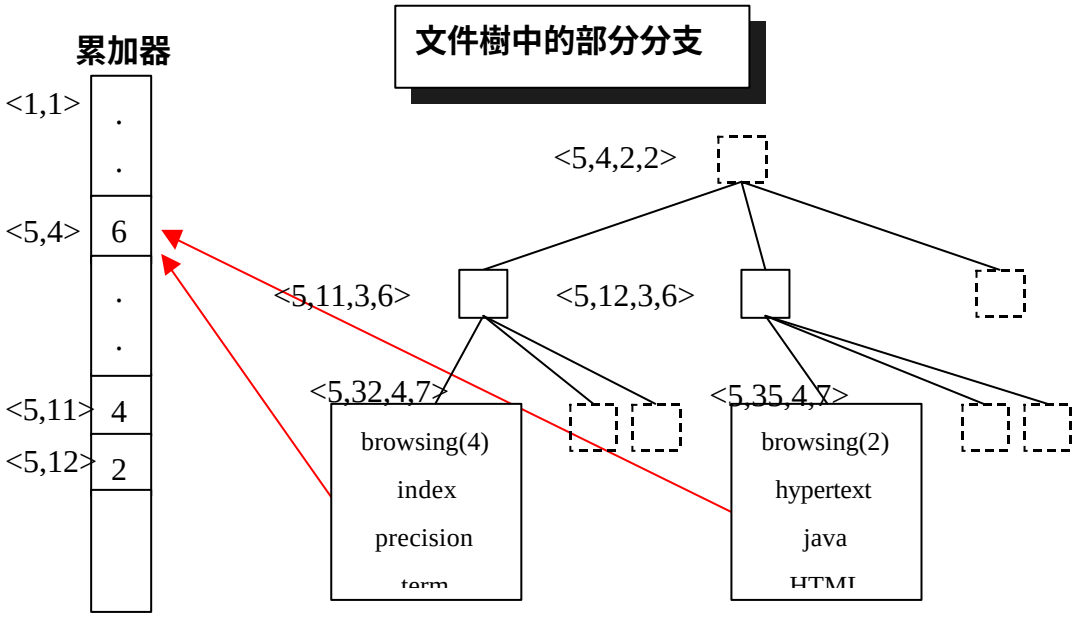


圖 4：由下而上的檢索機制

如圖 4 所示，每一個節點都有一組四個數值組成的 GID，其含意為<DID, UID, Level, Element >，Level 為該節點在樹狀結構中的深度，而 Element 則代表該節點的元素屬性代碼，元素屬性代碼係將結構化文件中的每一種元素都賦予一個數值編碼而成。以<5, 32, 4, 7>而言，表示此節點是在第五篇文件中 UID 為 32 的位置，且在樹狀結構中是位於第四層，其元素屬性代碼為 7。

有了 GID 的輔助，BUS 系統在執行檢索命令時，會從樹狀結構的底部往上搜尋。以圖 4 來看，首先系統會產生出一個累加器(Accumulators)來記錄檢索結果，假設欲檢索的關鍵字為 browsing，則系統一旦在節點中找到 browsing 這個字，便會將 browsing 出現的頻率累加到父節點的累加器中，以本例子即是<5,4>的累加器。系統按照此方法由下往上把整個樹狀結構尋找完後，所有關鍵字出現的頻率會記錄在累加器之中，系統再去尋找出現頻率最高的節點即表示是關聯性最高的節點。

綜合以上的說明，我們知道結構化文件索引的建置將會幫助檢索的進行，系統只需藉由索引即可快速地擷取到文件內的資料。然而在上述的相關研究工作中，只是純粹將結構化文件的結構資訊建置成索引，所能提供的僅僅是關鍵詞檢索，無法根據關鍵詞的語意來執行檢索動作，故本章第二節將介紹語意索引，一種可將詞鍵彼此間的語意作相互連結的索引方法。

## 第二節　語意索引的建置 (Semantic Indexing)

語意索引方法就是將兩兩詞鍵用語意相似性串連起來，進而建構成一個兩兩相連的網路脈絡，實際的呈現方法稱之為「語意矩陣(Semantic Matrix)」或「語意網路(Semantic Network)」。使用者可以藉由一個詞鍵來瀏覽網路脈絡中與之相似性高的其他詞鍵，比如說當使用者鍵入「數位圖書館」這個關鍵詞，系統也能檢索到「電子圖書館」、「內容管理」、「知識管理」、「網際網路」或是「資訊擷取」等關鍵詞。由於每個關鍵詞間皆有語意的關連性而非獨立存在，故語意索引的建置會建立起一個「概念空間」，透過語意索引的檢索，使用者所下達的檢索關鍵詞不僅僅只針對文字上的相似性作檢索，而是針對該檢索關鍵詞整個語意上的概念來作檢索。語意索引的優點即在於能正確地找出使用者感興趣的資訊，並附帶找出可能為使用者感興趣的資訊。

[Chung99]中提出了一個完整的語意索引建立方法，為了要有效地找出含有語意的關鍵字，在[Chung99]中使用了兩個關鍵的步驟，其一為名詞片語抽取(Noun Phrase Extraction)，主要目的為利用文字自動分析的技術將文件內含有語意的字詞分析出來；第二個步驟則為共同出現分析(Co-occurrence Analysis)，主要目的在於計算各詞鍵之間的關連性，有了共同出現分析的輔助，我們才能根據某一詞鍵的語意找出其他語意相近的詞鍵。

[Chung99]使用了 AZ Noun Phraser 來處理名詞片語抽取的步驟，整個流程分成下列三個階段：

1. 斷字(Tokenization)：這個階段主要的目的在於決定句子的分界，並將文句切分為個別分開的文字。在這個階段會將與語意無關的標點符號或是不具語意的文字消除，以增加名詞片語分析的正確性。
2. 詞類標記(Part-of-Speech Tagging)：此一階段包含「語彙分析(Lexical Analysis)」與「上下文分析(Contextual Analysis)」。前者會根據辭典中

**的語彙規則來為每一個斷字過的詞鍵作詞類標記；後者則是根據上下文的語意將意義含糊的詞類刪除掉，最後所留下的文字就可能成為一個有意義的名詞片語。**

3. **名詞片語辨識** (Noun Phrase Identification)**：這個階段根據名詞片語規則** (Noun Phrase Rules)**，以判斷經由前述階段處理後所留下來的文字是否符合一個名詞片語的詞性文法規則。若符合詞性文法規則，則判定其為一個富有語意的名詞片語。**

**經過以上三個階段的處理，就能將一份文件內所有含有語意的名詞片語抽取出來以作為詞鍵。接下來必須針對這些詞鍵作語意上的索引，在名詞片語的抽取過程中，我們尚須將每個名詞片語出現頻率的相關資訊都記錄下來，以便在共同出現分析中使用。**

**共同出現分析的主要功能就是將所有的詞鍵按照其語意，用網路的概念組織起來。共同出現分析利用** $tf_{ij}$**(詞鍵** $j$ **在文件** $i$ **中出現的頻率)與** $df_j$**(擁有詞鍵** $j$ **的文件數)來計算出詞鍵** $j$ **在文件** $i$ **中的權重，公式如下所示：**

$$d_{ij} = tf_{ij} \times \log(\frac{N}{df_j} \times w_j)$$

**方程式 3：詞鍵** $j$ **在文件** $i$ **中的權重公式**

**其中** $N$ **代表文件集中的文件總數，** $w_j$ **代表在詞鍵** $j$ **中的文字個數，在這個公式中可以發現一個詞鍵所含的文字個數越多，所計算出的權重也會越高，這是因為**[Chung99]**認為名詞片語由較多的文字組成即代表著較強的權重。計算出每一個詞鍵的權重之後，便可利用方程式 4 以計算出** $T_j$ **與** $T_k$ **之間的相似性。方程式中** $d_{ijk}$ **表示詞鍵** $j$ **與詞鍵** $k$ **同時出現在文件** $i$ **中的權重，** $df_{jk}$ **表示詞鍵** $j$ **與詞鍵** $k$ **同時出現的文件數量，** $N$ **所表示的是文件總數，** $w_j$ **則表示詞鍵** $j$ **的字元長度。**

**最後我們得到一個** T×T **的語意矩陣**(Semantic Matrix)**，**T **為詞鍵總數，而此矩陣中的每一個元素**(Element)**就是運用方程式 4 計算而得的權重，亦即代表詞**

鍵與詞鍵之間的相似性。這個方法的好處在於不僅僅能檢索到包含關鍵詞的文件，並能提供給使用者語意相似的關鍵詞以供查詢；相反地其缺點便在於建置語意索引時需要大量的數學運算，故可能需要平行處理的技術方可提升效能。

$$Weight(T_j, T_k) = \frac{\sum_{i=1}^{n} d_{ijk}}{\sum_{i=1}^{n} d_{ij}} \times WeightingFactor(T_k)$$

$$WeightingFactor(T_k) = \frac{\log \frac{N}{df_k}}{\log N}$$

$$d_{ijk} = tf_{ijk} \times \log(\frac{N}{df_{jk}} \times w_j)$$

**方程式 4：詞鍵相似性的計算公式**

本論文的目的，除了在結構化文件上作索引，並希望將文件內容納入語意索引的建置。本論文將於第三章中描述如何結合兩種索引，並試著將結構化文件中的階層式關係納入索引之中，以提供更強大更便利的檢索機制。

# 第三章　結構化文件索引與語意索引的結合

結構化文件索引可以幫助我們快速地檢索結構化文件，而語意索引則是讓使用者可以針對語意來作檢索，本文提出整合結構化文件索引與語意索引的想法，以期利用兩者之優點，帶給使用者更方便且適用的檢索機制。本章第一節首先概要描述系統架構及其運作流程，第二節則描述系統流程中各步驟的運作方式，說明如何將結構化文件中的階層式關係融入到語意索引之中，以提供更深層的檢索功能。

## 第一節　系統架構

本論文整合結構化文件索引以及語意索引，利用結構化文件的階層式樹狀結構特性再配合詞鍵與詞鍵語意上的相似性，發展出一個可藉由語意達到檢索目的之系統，系統架構如圖 5 所示：

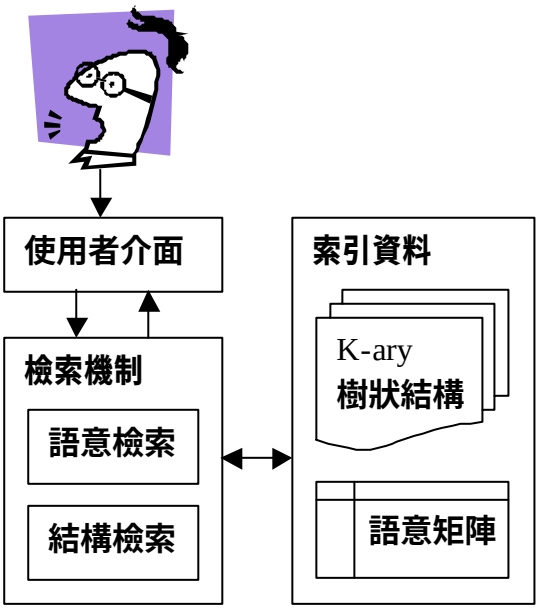


圖 5：系統架構

使用者可透過使用者介面輸入欲檢索的關鍵詞，接著檢索關鍵詞會被傳送至檢索機制。在檢索機制中會經過語意檢索與結構檢索的處理，配合索引資料中的 K-ary 樹狀結構與語意矩陣，找出符合使用者興趣的文件資料，更同時根據檢索

關鍵詞的語意找出在語意上相近的文件資料。

## 第二節 系統建置索引流程

系統建置索引流程圖如圖 6 所示：

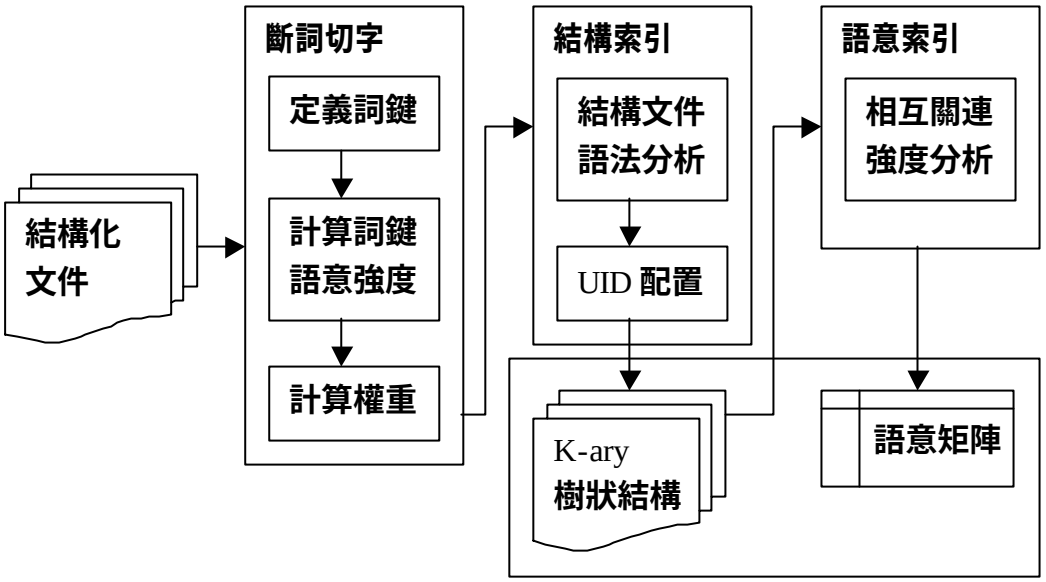


圖 6：系統索引建置流程圖

產生索引資料必須經過斷詞切字、結構索引以及語意索引三個步驟，斷詞切字的目的在於從文字資料中擷取出含有語意的詞鍵；結構索引則是為斷詞切字後的結構化文件建置索引；最後再將這些含有階層式架構的詞鍵交付語意索引步驟以找出詞鍵間相似性的語意矩陣。

在索引建置過程中產生的K-ary樹狀結構與語意矩陣即可提供給使用者作檢索，經由檢索機制與使用者所下的檢索關鍵詞作相似性比對，即可達到本文整合兩種索引的目的。下面我們便根據此三步驟逐一說明。

### 3.2.1 斷詞切字

由相關的研究工作可發現，在索引建置前必須先對原始的文件資料作斷詞切字之前置處理，才能產生出可供使用者檢索用的詞鍵。本文中的詞鍵長度均為二個中文字，也就是我們只針對雙字詞(Bi-gram)來作處理，這是因為在中文文件中雙字詞所能提供的檢索能力已經相當完整，即使我們再加入三字詞、四字詞甚至

五字詞，只會增加系統的運算時間而無法有效地提升檢索的查全率(Recall)及查準率(Precision)，故本文中略去三字詞以上的詞鍵。

詞鍵在文件中所出現的次數稱為詞鍵頻率(Term Frequency)，它會影響檢索的好壞。若建立高頻率詞鍵的索引會增加檢索的查全率，但卻會降低檢索的查準率。在中文字中有許多字的出現太過頻繁，不宜將之納入索引之中，因此在建立詞鍵索引時不予考慮，我們將這些文字稱作功能性文字(Function Words)。功能性文字在文件中出現次數大致是一樣多，但是非功能性文字在不同類型文件中出現的次數則會有差異。以「資訊」而言，是一個非功能性文字，它在資訊科學系或是資訊工程系的論文中出現的次數就可能會較多，而「論文」兩字為一個功能性文字，其出現次數無論在何種類型的論文中都可能差不多；故非功能性文字出現次數的多寡，即代表該詞鍵在文件中的重要性。根據詞鍵頻率的觀念，我們可依照下列步驟來產生建置索引所要用的詞鍵：

1. 讀入所有的中文字
2. 將所有功能性文字刪除
3. 依文字出現順序選出前後相鄰的中文字以組成詞鍵
4. 計算詞鍵頻率並計算其語意強度，語意強度代表的是一個由二中文字組成的詞鍵含有完整語意的強度，
5. 決定臨界頻率與臨界語意強度以篩選出欲納入索引的詞鍵

假設讀入的資料為「電腦的世界是多采多姿的。」，則系統會判別出「的」、「是」與「。」皆屬於功能性文字，而「電」、「腦」、「世」、「界」、「多」、「采」、「多」及「姿」為非功能性文字。接著系統由非功能性文字產生出候選詞鍵如下：「電腦」、「世界」、「多采」、「采多」與「多姿」。其中由於「腦」與「世」字中間被一個功能字隔開，故不列入於候選詞鍵中，同理可知「界多」亦不被列入。最後經過計算詞鍵頻率與語意強度並篩選之後，我們應可從「電腦的世界是多采多姿的。」文句中擷取出正確且語意強度夠高的詞鍵：「電腦」、「世界」、「多采」、與「多姿」。

語意強度可以利用下面的公式計算得到：

$$association(c1c1) = \frac{\frac{f_{c1c2}}{df_{c1c2}}}{\frac{f_{c1}}{df_{c1}} \times \frac{f_{c2}}{df_{c2}}}$$

**方程式 5：語意強度計算公式** [Kowalski97]

其中 *c1c2* 代表一個完整的雙字詞，*c1* 是第一個中文字，*c2* 則是第二個中文字。$f_{c1}$ 是指 *c1* 這個中文字在所有文件中出現的頻率，$df_{c1}$ 代表的就是這個中文字出現的文件總數，同理可得 $f_{c1c2}$ 就是代表著雙字詞在所有文件中出現的頻率，而 $df_{c1c2}$ 就是該雙字詞出現的文件總數。

我們試以表格 2 的雙字詞舉例說明之：

| 雙字詞 | *f* | *Df* | 字 | *f* | *df* | 字 | *F* | *df* |
|---|---|---|---|---|---|---|---|---|
| 向量 | 21 | 8 | 向 | 116 | 54 | 量 | 206 | 79 |
| 點對 | 11 | 5 | 點 | 277 | 70 | 對 | 372 | 117 |

**表格 2：計算雙字詞語意強度範例**

代入公式可得：

$$association(向量) = \frac{\frac{f_{向量}}{df_{向量}}}{\frac{f_{向}}{df_{向}} \times \frac{f_{量}}{df_{量}}} = \frac{\frac{21}{8}}{\frac{116}{54} \times \frac{206}{79}} = 0.469$$

$$association(點對) = \frac{\frac{f_{點對}}{df_{點對}}}{\frac{f_{點}}{df_{點}} \times \frac{f_{對}}{df_{對}}} = \frac{\frac{11}{5}}{\frac{277}{70} \times \frac{372}{116}} = 0.174$$

可以發現雖然「向」與「量」兩字出現的頻率比「點」與「對」出現的頻率較低，但「向量」兩字組合起來出現的頻率卻高於「點對」兩字的頻率，因此我們可得知雙字詞「向量」含有較強的語意強度，經由公式所計算出來的語意強度也的確符合我們的假設。

計算完各詞鍵的語意強度後，每一篇文件都會產生許多語意夠強的詞鍵，但並非每一個詞鍵都足以代表該文件，尚須根據每篇文件中各詞鍵權重高低來選擇

足夠代表該文件的詞鍵。我們利用傳統資訊擷取的公式來作詞鍵在文件中的權重計算，公式如下：

$$w_{ij} = tf_{ij} \times \log \frac{N}{df_j}$$

**方程式 6：詞鍵在文件中的權重計算公式** [Kowalski97]

其中 $w_{ij}$ 代表詞鍵 $j$ 在文件 $i$ 中的權重，權重越高表示該詞鍵越能代表該文件的內容。$tf_{ij}$ 代表詞鍵 $j$ 在文件 $i$ 中出現的次數，$df_j$ 則是出現詞鍵 $j$ 的文件數量。$tf_{ij}$ 代表的意義是提高出現次數較多的詞鍵權重，但考慮到有些詞鍵是很常見且並非對該文件有特別意義，故加入了 $\log \frac{N}{df_j}$ 來減低在很多文件中皆出現的詞鍵權重。

綜合以上所述，在經過斷詞切字後，每一個文件皆可找出足以代表文件內容且語意夠強的詞鍵，稱為文件描述子(Document Descriptor)。當我們針對文件下檢索關鍵詞時，即可根據這些文件描述子來找到感興趣的文件資料，不需要再根據全文資料去作檢索。

### 3.2.2 結構索引

```
<?xml version="1.0" encoding="big-5"?>
<?xml-stylesheet type="text/xsl" href="show_book.xsl"?>
<叢書>
    <書籍>
        <作者>張三</作者>
        <書名>Linux 白皮書</書名>
        <出版年>1999</出版年>
    </書籍>
</叢書>
```

**圖 7：XML 文件範例**

處理完斷詞切字的程序後，接著必須將文章的階層式架構轉換成文件樹狀結構(Document Tree Structure)，最後利用 K-ary Tree 的概念將每一個分析出來的節

點皆配置一個 UID。我們使用微軟的 XML Parser 來作為結構化文件的結構分析工具，XML Parser 可分析出結構化文件的節點資訊，以圖 7 之 XML 文件為例，XML Parser 會將該結構化文件展開成圖 8 之樹狀結構圖，其中以方框框起來的表示資料而非節點資訊。

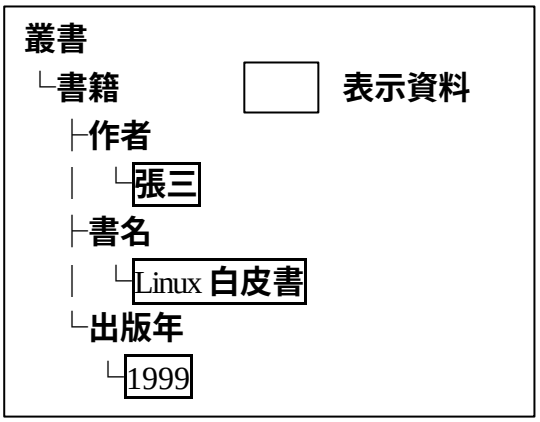


圖 8：XML 樹狀結構圖

在此一 XML 文件中，XML Parser 根據結構化文件階層式的特性將每一個節點都分析出來並存放於一個樹狀結構中。如圖 8 所示，「叢書」是整個樹狀結構的根節點(Root Node)，「書籍」則是其子節點，同樣地「書籍」本身亦擁有多個子節點，每一個子節點則分別擁有不同的資料。

```
CurrentLevel = 0;
CurrentBrahch = 0;
Function Traverse(Node p){
    IF (ChildExist(p) && NotVisited) {  // 檢查是否存在尚未配置過的子節點
        CurrentLevel ++;
        CurrentBranch++;
        // 根據 CurrentBranch 跟父節點的 UID 配置此節點之 UID
        p.UID =  MaxBranch*(p.parent->UID-1)+CurrentBranch+1;
        Traverse(p.child[CurrentBranch]);  // 往下處理第 CurrentBranch 個子節點資訊
    } ELSE
        Traverse(p.parent);  // 跳回父節點繼續處理
}
```

圖 9：UID 配置演算法

藉由 XML Parser 的分析，可得到一個文件樹狀結構(Document Tree

Structure)，我們再根據深度優先搜尋法(Depth First Search)瀏覽整個樹狀結構並配置 UID，將原始的樹狀結構轉換成為 K-ary Tree，即可完成結構化文件之索引。以圖 7 之文件為例，首先得知其最大分支數為 3，故必須將之對應至一個 3-ary Tree 上面。利用深度優先搜尋法配置 UID 的演算法如圖 9 所示。

根據 K-ary Tree 的特性，只要知道本身節點的 UID 即可計算子節點的 UID，公式為 $child(i, j) = k(i-1) + j + 1$，其中 $i$ 表示此節點的 UID，$j$ 表示欲計算第 $j$ 個子節點的 UID，而 $k$ 表示最大的分支個數。以<作者>節點為例，欲配置<作者>節點之 UID，則必先取得父節點<書籍>之 UID 為 2，且<作者>為<書籍>節點的第一個分支節點，故<作者>節點之 UID 為 3(2-1)+1+1=5。以圖 7 為例，最後配置完成的 UID 如表格 3 所示。

| 節點 | UID |
|---|---|
| 叢書 | 1 |
| 書籍 | 2 |
| 作者 | 5 |
| 張三 | 14 |
| 書名 | 6 |
| Linux 白皮書 | 17 |
| 出版年 | 7 |
| 1999 | 20 |

表格 3：UID 配置結果

配置完成 UID 的動作後，必須將所有分析過的文件資料儲存起來，我們使用的資料結構為(DID, UID, Level, Element Type, Content)。以圖 7 為例，最後真正儲存成索引的資料如表格 4 所示：

| DID | UID | Level | Element Type | 節點 |
|---|---|---|---|---|
| 1 | 1 | 1 | Root | 叢書 |
| 1 | 2 | 2 | Element | 書籍 |
| 1 | 5 | 3 | Element | 作者 |
| 1 | 14 | 4 | Text | 張三 |
| 1 | 6 | 3 | Element | 書名 |
| 1 | 17 | 4 | Text | Linux 白皮書 |
| 1 | 7 | 3 | Element | 出版年 |
| 1 | 20 | 4 | Text | 1999 |

表格 4：最後儲存索引

### 3.2.3 語意索引

傳統的語意索引方法將兩兩詞鍵用語意相似性串連起來，進而建構成一個兩兩相連的網路脈絡，實際的呈現方法稱之為「語意矩陣(Semantic Matrix)」或「語意網路(Semantic Network)」。使用者可以藉由一個詞鍵來瀏覽網路脈絡中與之相似性高的其他詞鍵。以圖 10 為例，以「數位圖書館」為中心，可找出較相關的幾個詞鍵：「電子圖書館」、「內容管理」、「資訊擷取」、「知識管理」以及「網際網路」。

在圖 10 中由「數位圖書館」指出去的箭號代表著和「數位圖書館」語意相關之其他詞鍵的權重，權重越高則表示其語意關連性越強，值得注意的是，在語意相似網路中箭號是有方向性的，亦即「數位圖書館」相對於「資訊擷取」的權重，並不一定等於「資訊擷取」相對於「數位圖書館」的權重。

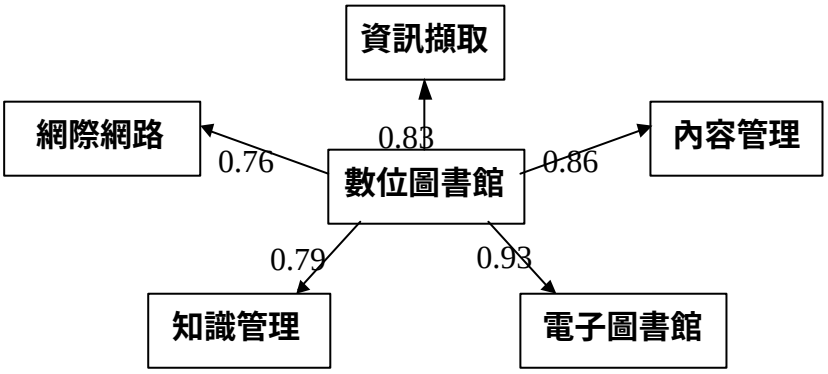


圖 10：語意關連性示意圖

傳統語意索引的概念可以幫助使用者找到文件中詞鍵與詞鍵的相似性，應用在結構化文件上亦可得到相同的功能，但在傳統文件中建置索引的方法乃是利用詞鍵在文件中出現的頻率以及出現的文件總數做權重計算，沒有將結構化文件的階層式資訊考慮進去，當使用者下了一個檢索關鍵詞，系統只能找出與該檢索關鍵詞語意相近的「文件」，而非在找出語意相近的「元素」。

有鑑於此，本文提出一個新的權重計算方式將結構化文件的階層式資訊考慮進來。在斷詞切字的處理流程中，我們不儲存詞鍵出現在文件中的頻率，改成儲

存詞鍵在每一個元素中出現的頻率，並在計算權重的地方做了些調整。原始相似性權重計算的公式如下：

$$Weight(T_j,T_k)=\frac{\sum_{i=1}^{n} d_{ijk}}{\sum_{i=1}^{n} d_{ij}}\times WeightingFactor(T_k)$$

**方程式** 7**：原始相似性權重計算公式** [Chung99]

**展開公式如方程式** 8 **所示**:

$$Weight(T_j,T_k)=\frac{\sum_{i=1}^{n} d_{ijk}}{\sum_{i=1}^{n} d_{ij}}\times WeightingFactor(T_k)=\frac{\sum_{i=1}^{n} tf_{ijk}\times\log(\frac{N}{df_{jk}}\times w_j)}{\sum_{i=1}^{n} tf_{ij}\times\log(\frac{N}{df_j}\times w_j)}\times\frac{\log(\frac{N}{df_k})}{\log N}$$

**方程式** 8**：原始相似性權重計算公式展開**

**從方程式** 8 **中可以發現** *tf* **與** *df* **皆是根據詞鍵與文件的關係得知，而我們斷詞切字處理過程中卻希望所儲存的資料是根據詞鍵與元素的關係，故定義** $tfe_{ij}$ **與** $ef_j$ **如下所述：**

1. $tfe_{ij}$(Term Frequency at Element)**：表示在元素** *i* **中詞鍵** *j* **出現的頻率。**
2. $ef_j$(Element Frequency)**：表示詞鍵** *j* **出現的元素總數。**

**有了上述兩個定義，當我們欲計算詞鍵在元素中與其他詞鍵的相似性權重時，則可直接轉換原始權重計算公式如方程式** 9 **所示：**

$$Weight(T_j,T_k)=\frac{\sum_{i=1}^{n} d_{ijk}}{\sum_{i=1}^{n} d_{ij}}\times WeightingFactor(T_k)=\frac{\sum_{i=1}^{n} tfe_{ijk}\times\log(\frac{N}{ef_{jk}}\times L_j)}{\sum_{i=1}^{n} tfe_{ij}\times\log(\frac{N}{ef_j}\times L_j)}\times\frac{\log(\frac{N}{ef_k})}{\log N}$$

**方程式** 9**：套用元素中詞鍵頻率的相似性計算權重的公式**

**我們所設計的方程式** 9 **，不使用原本權重計算公式中的** $w_j$**，公式中** $w_j$ **係指意義為詞鍵的字元長度，也就是希望為字元長度較長的詞鍵增加較高的權重，但本文中所討論的詞鍵皆為雙字詞，每一個詞鍵的字元長度皆為** 2**，故無法看出** $w_j$ **的好處。本文將** $w_j$ **代換為** $L_j$**，** $L_j$ **所代表的即是詞鍵** *j* **在** K-ary **樹狀結構中的深**

度。樹狀結構中越上層的元素代表著越廣泛的範圍，越底層的元素則會具有較精確的範圍，故我們以半衰期來計算 $L_j$ 的數值，本文提出方程式 10 來作為半衰期的計算公式，*depth* 為整個 K-ary Tree 的總深度，而 *depth(j)*則表示節點 *j* 在樹狀結構中的深度。

$$L_j = \frac{2^{depth(j)-1}}{2^{depth}-1} \quad \text{where } 1 \le \text{depth(j)} \le \text{depth}$$

方程式 10：半衰期計算公式

以圖 7 為例，分別計算出每一個節點的 $L_j$ 如表格 5 所示，位於最底層的節點之 $L_j$ 值為 8/15，而位於最上層的節點僅有 1/15，如此即可將階層式結構上下層元素的關係包含在語意索引之中。

| 節點 | Level | $L_j$ |
|---|---|---|
| 叢書 | 1 | 1/15 |
| 書籍 | 2 | 2/15 |
| 作者 | 3 | 4/15 |
| 張三 | 4 | 8/15 |
| 書名 | 3 | 4/15 |
| Linux 白皮書 | 4 | 8/15 |
| 出版年 | 3 | 4/15 |
| 1999 | 4 | 8/15 |

表格 5：$L_j$ 數值表

有了 $L_j$ 的計算方式，我們即可代入方程式 9 中計算詞鍵的相似性權重。此外，在方程式中所用到的 *tfe* 與 *ef* 是詞鍵在元素中的出現頻率以及詞鍵出現的元素個數，故計算出來的相似性是詞鍵相對於元素的權重，而非相對於整個文件的權重。若使用者所下的檢索關鍵詞僅希望檢索到最上層的資料，也就是說只需要提供文件的純文字內容供檢索，而不需包含節點資訊，則我們必須使用方程式 8 來作相似性權重計算。此時會衍生出一個問題：*tfe* 與 *ef* 的元素層(Element Level)資訊是否能由下而上地還原且滿足文件層(Document Level)的資訊？

由方程式 8 中可得知欲還原文件層的資訊，則公式中必須使用到 *tf* 以及 *df*，我們試著以表格 6 為例，利用 *tfe* 和 *ef* 來還原 *tf* 以及 *df*。

| | 文件 1 | | | | 文件 2 | | |
|---|---|---|---|---|---|---|---|
| | **E1** | **E2** | **E3** | **E4** | **E1** | **E2** | **E3** |
| **詞鍵 1** | 0 | 2 | 1 | 0 | 3 | 6 | 0 |
| **詞鍵 2** | 0 | 0 | 0 | 2 | 0 | 0 | 1 |

**表格 6：詞鍵頻率表格**

**由表格 6 中可看出詞鍵 1 在文件 1 的各元素中出現的頻率分別為 0、2、1 以及 0，故可以很快地累加得出詞鍵 1 在文件 1 出現的頻率為 0+2+1+0=3，同理可得詞鍵 2 在文件 1 中的出現頻率為 2。我們只需要利用簡單的計算即可將「元素中詞鍵的出現頻率(*tfe*)」還原至「文件中詞鍵出現頻率(*tf*)」。**

**詞鍵之文件頻率累加演算法如圖 11 所示，我們利用一累加器來計算「出現詞鍵的文件數量(*df*)」，針對每一篇文件的每一個元素中詞鍵的出現頻率作檢查，若出現頻率大於 0 則累加器加 1，並跳至下一篇文件作處理。以表格 6 為例，欲計算詞鍵 1 出現的文件頻率，則首先檢查文件 1 中的每一個元素，因為 E1 的出現頻率為 0，故繼續檢查下一個元素，在 E2 時即發現頻率為 2，表示在該文件中詞鍵 1 有出現過，故 *df* 累加 1，並跳出迴圈。接著在文件 2 中依此方法檢查，發現詞鍵 1 在 E1 的出現頻率為 3，表示在文件 2 中詞鍵 1 也有出現過，故 *df* 亦累加 1 變成 2。最後我們得知詞鍵 1 的 df 為 2，即表示詞鍵 1 的「詞鍵出現的文件頻率」為 2。**

```
Function Accumulate(){
    For EachDocument   // 檢查每一個文件
        For EachElement { // 檢查文件中每一個元素
            IF (tfe > 0 ) {  // 檢查詞鍵的元素頻率是否非 0
                df++;
                BREAK;
            }
        }
    }
}
```

**圖 11：詞鍵之文件頻率累加演算法**

本文提出儲存詞鍵在元素中出現頻率的方法，不僅能計算詞鍵相對於元素的權重，也能很快速地計算出詞鍵相對於整個文件的權重。只需要簡單的加法運算，便可以將詞鍵檢索的深度從元素層(Element Level)轉換至文件層(Document Level)，大大地加強了原本語意索引僅能處理的資訊範圍。

# 第四章　系統實作與結果分析

本章將簡介本論文實作系統並針對本論文所提出結構化文件索引與語意索引進行效益的評估與分析比較。第一節簡介本文之實作系統，第二節說明評估檢索結果所採用的方式和標準，第三節是利用本論文之索引方法進行索引建置的效益評估並討論，第四節則是利用本論文之索引方法進行關鍵詞檢索的評估結果。

## 第一節　實作系統

本文開發出一個供使用者檢索的 Web-based 系統，使用者可透過瀏覽器執行檢索的動作，輸入檢索關鍵詞並指定欲檢索的元素名稱，系統即可執行檢索的動作以找出相符合的資訊。系統介面圖如

圖 12 所示。Keyword 是欲檢索的關鍵詞，而 Field 則是欲檢索的元素名稱，使用者可以選擇任一元素來作檢索。本系統的資料來源是以民國 88 年國立交通大學博碩士論文之 XML 格式書目與摘要資料為基礎，總共有 1219 篇論文，每一篇論文的書目資料大小約為 10K，總大小為 13.1 MB，內容包含有每篇論文的書目資料以及摘要。建置索引的環境硬體配備為 Intel Petium III 500 的 CPU，系統記憶體有 192 MB；作業系統為 Windows 2000 Professional，使用 Borland C++ Builder 5.0 企業版做為軟體發展工具。

以結構化索引的檢索為例，若使用者欲檢索的資訊為「在 Subject 元素中擁有影像關鍵詞的資訊。」則必須在 Keyword 欄位中填入「影像」，在 Field 欄位中填入「Subject」；則系統將在執行檢索完後回傳給使用者所有符合的節點資訊，結果畫面如圖 13 所示。系統會列出 DID、UID、Title 與 Element，並提供超連結以便讓使用者可以檢閱詳細的文件內容。DID 表示文件的編號，UID 是符合使用者輸入關鍵詞的節點 UID，Title 是該文件的標題，Element 則是該節點的元素名稱。其中本系統為了測試系統之成效，亦增加了結果回饋機制供使用者勾選；

使用者可根據系統檢索結果與原始檢索關鍵詞作比對，勾選出符合使用者興趣之文件，最後送出給系統供統計之用。

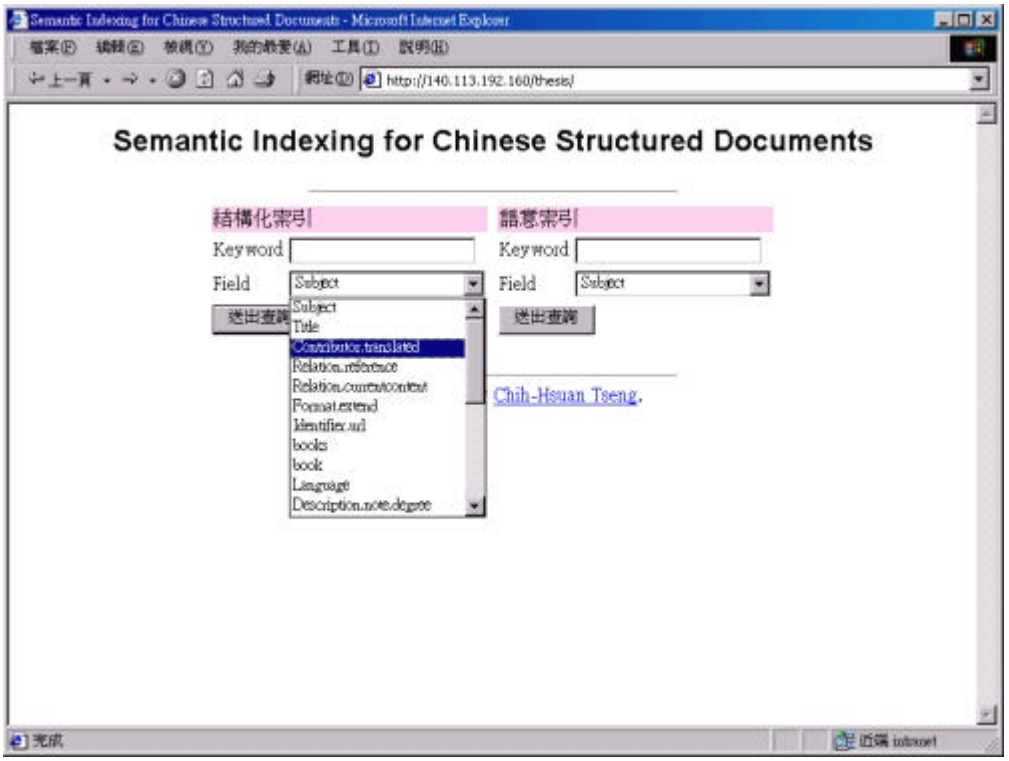


圖 12：系統介面

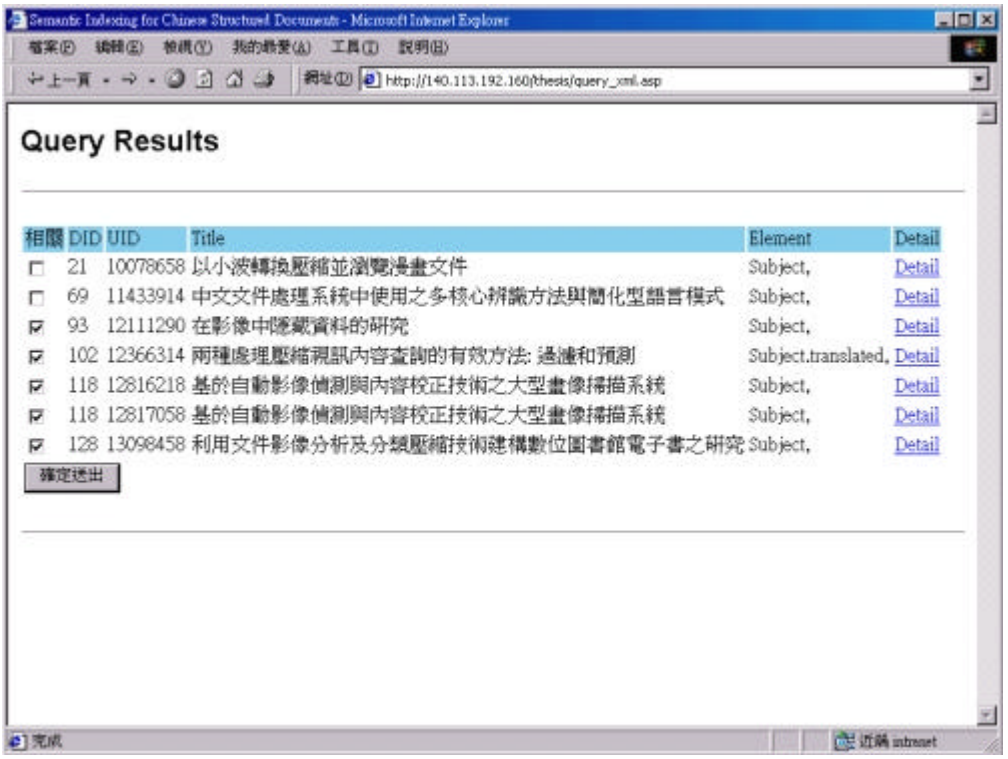


圖 13：結構化索引之檢索結果畫面

同樣地，以語意索引之檢索為例，輸入相同的關鍵詞與元素型態，則表示使用者感興趣的資訊為「在 Subject 中含有影像語意的文件資訊。」在語意索引中，系統不僅會檢索「影像」，並一併檢索與影像語意相關的詞鍵，檢索結果畫面如圖 14 所示，很明顯地可以看出擷取出的資訊會比結構化索引的結果多，故語意索引在求全率方面會遠比結構化索引來得好。

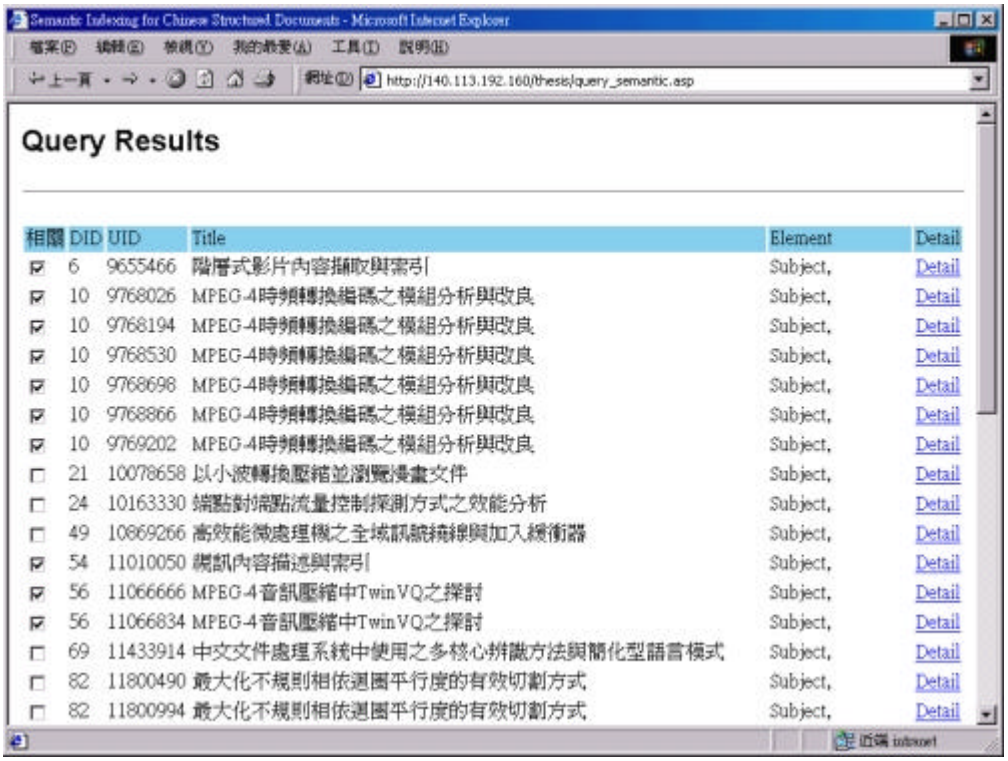


圖 14：語意索引之檢索結果畫面

## 第二節　效益評估方法

資訊檢索系統一般的效益評估方法多是以查準率(Precision)與查全率(Recall)來作為評斷的標準，查準率和查全率的定義為：

$$precision = \frac{\#\ of\ \ retrieved\ \ and\ relevant\ items}{\#\ of\ \ retrieved\ items}$$

$$recall = \frac{\#\ of\ \ retrieved\ and\ relevant\ items}{\#\ of\ \ relevant\ items}$$

其中擷取出且相關的文件數量(# of retrieved and relevant items)是由使用者直接檢視系統回傳的資料，找出語意上認定是符合使用者檢索需求的結果；所有擷取出的文件數量(# of retrieved items)是系統傳回的所有文件總數；所有相關的

文件數量(# of relevant items)為原始資料中真正符合使用者檢索需求的文件數量。查全率是計算在相關文件中經由檢索被擷取出來的文件比例，而查準率則是計算在檢索回傳的結果中為相關文件的比例。

## 第三節 索引建置之效益評估

本節分別討論結構化文件索引與語意索引兩種方法，在建置索引時所需花費的時間及空間，並探討索引建置的成效。

### 4.3.1 索引建置時間之效益評估

首先我們根據不同的原始資料大小來評估索引建置的時間。以交大博碩士論文之書目與摘要資料為基礎，並抽取出部分論文當作比較的對象。藉由改變測試資料量的大小來觀察是否會影響建置索引的時間，並分別根據結構化文件索引及語意索引做評估，以比較兩者的效益。

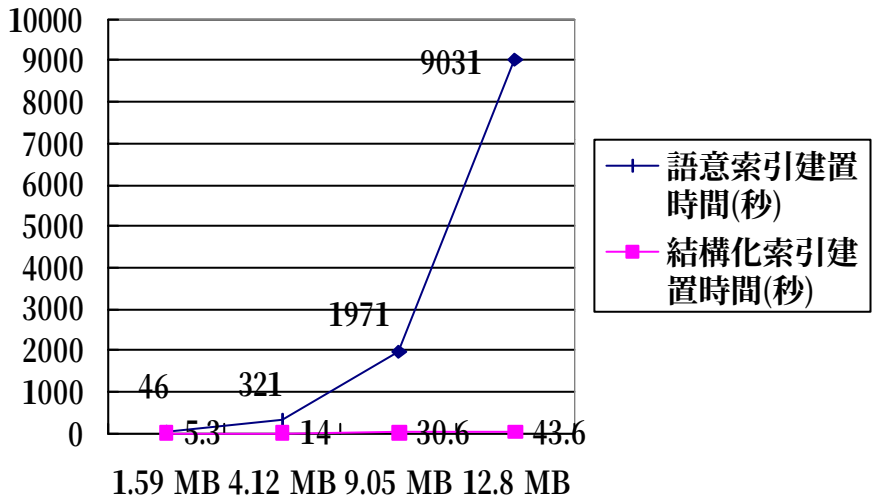


圖 15：改變測試資料大小之索引建置時間

圖 15 所代表的是改變資料量所得之結構化文件索引建置時間與語意索引建置時間，分別取出測試資料大小 1.59 MB、4.12 MB、9.05 MB 以及 12.8 MB 來進行比較，發現建置結構化索引的時間隨著測試資料大小成線性成長，而不是成指數成長(Exponential)。當測試資料大小倍增時，建置結構化索引的時間也會

隨著線性成長，不會成指數成長，故即使將本論文提出的演算法用於龐大的資料時，也不必擔心建置結構化索引的時間會急遽增加。

然而，觀察語意索引的成長曲線，可發現當測試資料量變大時，語意索引建置的時間會隨著遽增。

```
Function SemanticIndexing(){
    For I = 1 to MaxTerm { // 檢查每一個詞鍵做動作
        For J = 1 to MaxTerm { // 檢查每一個詞鍵做動作
            WeightCaculation(I, J);
        }
    }
}
```

圖 16：語意索引之演算法

圖 16 為語意索引之部分演算法，從該演算法可以推論：假設原始資料中唯一的詞鍵總個數為 N，則建置語意索引的時間約為 $N^2$；故當詞鍵總個數倍增為 2N 時，則建置語意索引的時間將增加為 $(2N)^2$，為原本的 4 倍。從圖 15 中也可以發現確實符合該推論，但由於程式在原始資料大小為 4.12 MB 以上時，實體記憶體不敷使用，必須挪用硬碟來作為虛擬記憶體，在執行效率上會大受影響，故在實作上的語意建置索引時間約為 6 倍。

綜觀上述結構化文件索引與語意索引的建置時間比較，可以發現結構化文件之索引相當適用於大量結構化文件的索引，即使資料量大幅度地增加，結構化索引也能快速地建置索引；然而，語意索引則因為所需花費的計算時間過於龐大，應用於大量文件的索引時稍嫌不足，這也是[Chung99]將平行計算納入語意索引中的原因。

### 4.3.2 索引建置空間之效益評估

圖 17 所示即為建置索引所需之空間，當原始資料大小增加時，結構化索引也會隨著大量增加，且所需的儲存空間亦遠比語意索引大。這是因為結構化索引

必須將文件的結構資訊也納入索引的考量中，故原始資料量越大，所需的索引空間也越大。以表格 4 為例，最後所儲存的索引不但保持了原始的資料，且為了提供結 Structure Query 的功能，亦必須保留 DID、UID 以及元素型別(Element Type)，故最後的索引儲存空間會隨著資料量增加而大增，也遠比語意索引多。

另一方面，語意索引需要的儲存空間即為整個「語意矩陣」。語意矩陣的大小與原始資料中的詞鍵個數有關，當詞鍵數量增加，語意矩陣便會增加，導致語意索引的空間也增加。然而在本論文中，為了不讓詞鍵的個數過度膨脹，我們最後儲存的語意矩陣索引僅存入每個詞鍵的前 10 個相似性權重最高的詞鍵，不但能提供給使用者較精確的結果，亦能有效地減少了語意索引的儲存空間。

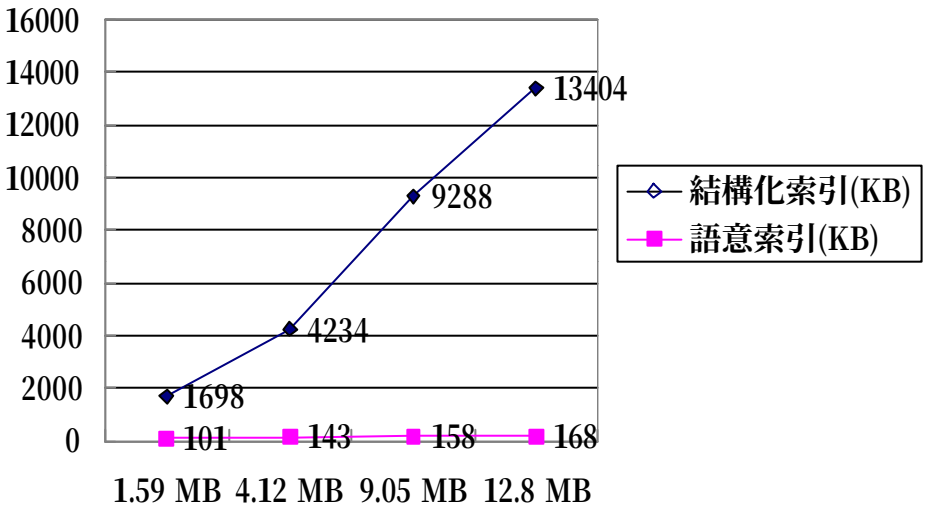


圖 17：建置索引所需空間

### 4.3.3 索引建置之成效

我們挑選出文件中語意強度較強的前 30 個詞鍵以及詞鍵權重較高的前 30 名來檢視索引建置之成效。

| | 語意強度 | | 詞鍵權重 | |
|---|---|---|---|---|
| 編號 | 強度 | 詞鍵 | 權重 | 詞鍵 |
| 1 | 0.999701 | 民國 | 1.000000 | 影像 |
| 2 | 0.979167 | 致謝 | 0.633224 | 網路 |
| 3 | 0.926959 | 樣本 | 0.614711 | 視訊 |
| 4 | 0.881288 | 細度 | 0.594361 | 行動 |
| 5 | 0.859091 | 相依 | 0.578215 | 通訊 |
| 6 | 0.818591 | 擴充 | 0.557606 | 壓縮 |
| 7 | 0.808421 | 困難 | 0.552619 | 指令 |
| 8 | 0.797517 | 精細 | 0.535699 | 協定 |
| 9 | 0.750000 | 註冊 | 0.532895 | 服務 |
| 10 | 0.728615 | 概念 | 0.532895 | 學習 |
| 11 | 0.719801 | 必須 | 0.509868 | 模型 |
| 12 | 0.717197 | 碩士 | 0.492986 | 辨識 |
| 13 | 0.717197 | 首先 | 0.486269 | 預測 |
| 14 | 0.712251 | 身份 | 0.483638 | 分類 |
| 15 | 0.688877 | 博士 | 0.480263 | 影片 |
| 16 | 0.678386 | 說明 | 0.475835 | 精細 |
| 17 | 0.674419 | 希望 | 0.464019 | 個人 |
| 18 | 0.663310 | 連續 | 0.454456 | 管理 |
| 19 | 0.656566 | 世界 | 0.454017 | 物體 |
| 20 | 0.656074 | 工具 | 0.449013 | 記憶 |
| 21 | 0.654545 | 智慧 | 0.433661 | 模組 |
| 22 | 0.654144 | 範圍 | 0.431524 | 網站 |
| 23 | 0.652578 | 並行 | 0.431428 | 輪廓 |
| 24 | 0.652016 | 座標 | 0.430431 | 訊協 |
| 25 | 0.637352 | 減少 | 0.427632 | 機制 |
| 26 | 0.636824 | 外型 | 0.415359 | 路徑 |
| 27 | 0.636364 | 門檻 | 0.412404 | 自動 |
| 28 | 0.634615 | 修正 | 0.409539 | 設計 |
| 29 | 0.628045 | 根據 | 0.409204 | 人化 |
| 30 | 0.621951 | 神經 | 0.403900 | 搜尋 |

**表格 7：前 30 名的語意強度以及權重強度**

**表格 7 的詞鍵是從資訊科學所與資訊工程所之博碩士論文建置索引後之結果中，挑選出某篇文件中語意強度及詞鍵權重較高之前 30 名，我們可以從中可以發現，語意強度高的詞鍵並不一定表示詞鍵的權重高，語意強度高的詞鍵通常代表著較常出現且可能為一個有語意的雙字詞；相反地，詞鍵強度高的詞鍵則表**

示該文件之內容與該該詞鍵之語意概念相接近。由於本文中所使用的詞鍵皆為雙字詞，故在挑選詞鍵的過程中可能會發現有些並不足以構成語意的詞鍵亦被選出。如詞鍵權重表中編號 29 的「人化」，我們可以認定該詞鍵為一個不具有語意的雙字詞，然而對照編號 17 的「個人」並探究原因，可發現其實真正需要被挑選出的詞鍵應為「個人化」，因為「個人化」的語意強度不但夠，且詞鍵權重亦夠高，符合被選出的條件，但由於本文僅止於討論雙字詞的關係，故無法將之納入詞鍵中，造成語意遺失的情形。然而檢視其餘高權重的詞鍵，發現仍相當足以代表文件的概念及內容，並足以提供為檢索之用。故這些語意遺失的情形我們予以忽略不計。

在語意索引的建置成效方面，我們挑出五個詞鍵來作為說明，每個詞鍵後面列出與其相似性權重最高的五個相似詞鍵，括弧內則為其相似權重，此即為語意矩陣中的一小部分，詳細資料如表格 8 所示。

| 詞鍵 | 相似詞鍵 1 | 相似詞鍵 2 | 相似詞鍵 3 | 相似詞鍵 4 | 相似詞鍵 5 |
|---|---|---|---|---|---|
| 影像 | 訊號(5.651250) | 延遲(5.326275) | 索引(5.273580) | 編碼(4.968718) | 雜訊(4.130671) |
| 編碼 | 時間(3.134810) | 架構(3.129864) | 效能(3.126399) | 設計(2.987573) | 雜訊(1.478415) |
| 訊號 | 技術(3.123744) | 網路(3.126132) | 分析(2.686237) | 模擬(2.600478) | 雜訊(1.271184) |
| 視訊 | 模擬(4.133471) | 整合(3.968198) | 分割(3.911765) | 訊息(3.907350) | 輪廓(2.193699) |
| 雜訊 | 網路(4.034743) | 應用(3.393450) | 環境(3.317864) | 設計(3.221039) | 雜訊(0.619790) |

表格 8：部分語意矩陣

以「影像」為例，分別有「訊號」、「延遲」、「索引」、「編碼」與「雜訊」五個相似詞鍵，此六個詞鍵之間的相似性便構成一個「語意空間」或「概念空間」。當使用者欲檢索「影像」此一語意時，系統除了會檢索「影像」詞鍵外，並會根據語意矩陣表中找出五個相似詞鍵，分別根據此五詞鍵做檢索，最後再將結果整合回傳給使用者，這就是語意索引以「語意矩陣」及「概念空間」的觀念代替傳統的關鍵詞檢索的方式。

## 第四節 關鍵詞檢索之效益評估

在此節中將分別評估結構化文件索引與結合結構化文件索引與語意索引之方法(以下稱本論文索引方法)，利用此兩種索引進行關鍵詞檢索的評估。

考慮使用者對回傳結果的熟悉程度，我們取出資訊科學所與資訊工程所之博碩士論文書目與摘要資料當作效益評估的測試資料，並請十位相關科系之使用者輸入五組不同的關鍵字，分別為「影像」、「網路」、「視訊」、「網站」與「壓縮」，再由使用者挑選主觀認定為語意上相關的文件作為「擷取出且相關的文件數量」，代入查全率與查準率之公式做計算。

圖 18 為利用結構化索引與本論文索引方法進行關鍵詞查詢的效益比較。我們可以發現結構化索引會比本論文索引的查準率還高。

結構化文件索引建置的資料除了節點資訊外，並會記錄下完整的文件內容資訊，因此根據使用者所輸入的關鍵詞做檢索，所得之結果必符合檢索關鍵詞，故查準率會較高；而本論文之索引不僅找出符合使用者輸入之關鍵詞，並根據語意矩陣內相似性權重高的詞鍵來做檢索，故檢索所得之文件資料會較多，導致查準率的分母增加，故會降低查準率。

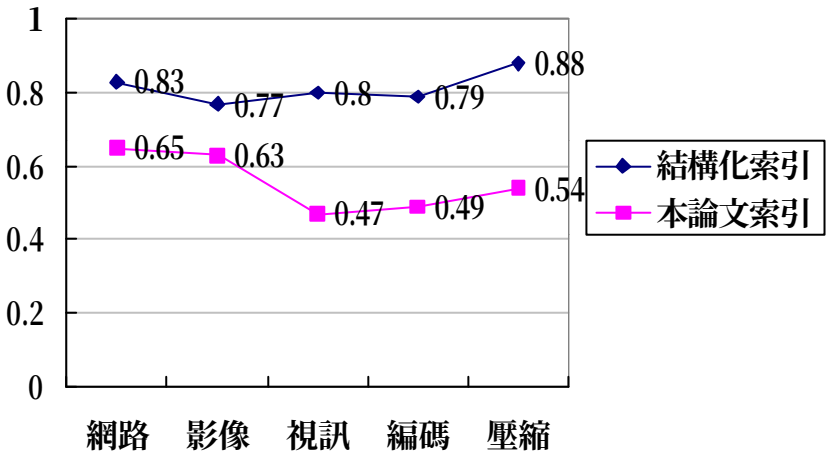


圖 18：查準率之效益評估

圖 19 則為查全率之效益比較，由於本論文索引可將關鍵詞轉為語意空間的概念來呈現，故系統除了會得到含有關鍵詞的文件資訊外，還會根據相關性權重高的詞鍵擷取出語意上可能有相關的文件資訊，因此會增加查全率的分子，導致查全率變高。因此我們可發現本論文索引方法之查全率會優於結構化索引之方法，也就是說本論文之索引方法會比傳統的結構化索引方法更能找到使用者感興

趣的資訊。

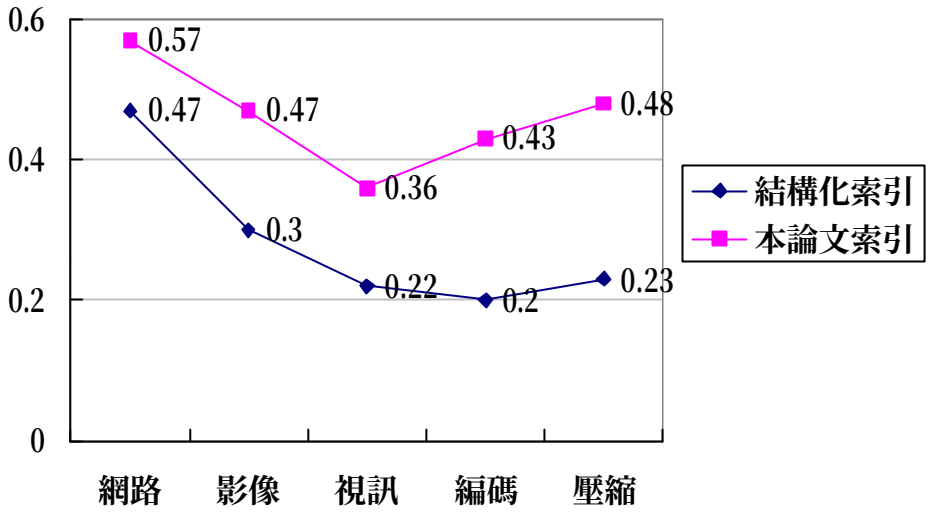


圖 19：查全率之效益評估

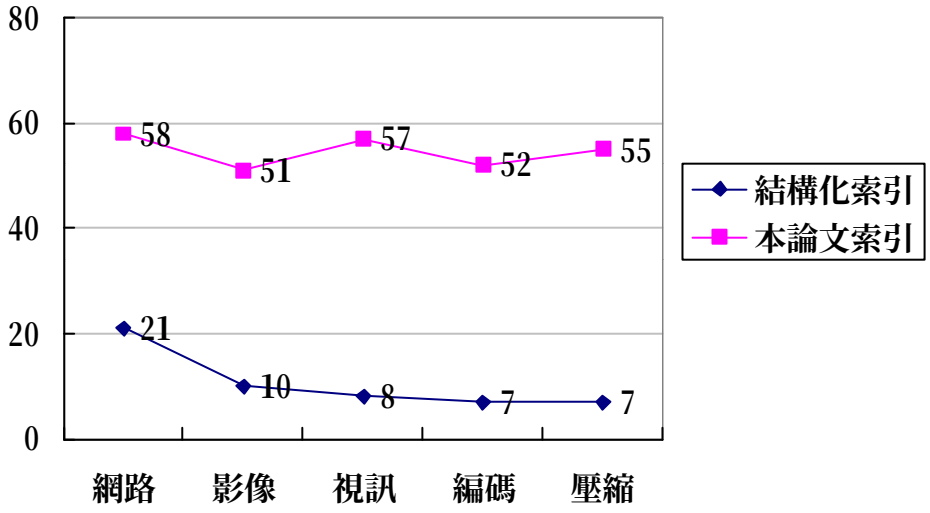


圖 20：查詢處理時間

圖 20 則是處理查詢的執行時間，可以發現本論文之方法較耗時，這是由於結構化索引只需針對使用者輸入之關鍵詞做檢索，不像本論文之索引方法尚會根據語意相關的詞鍵做檢索，若系統設定一併檢索 10 個相關性高的詞鍵，則必須根據各詞鍵檢索 10 次，故查詢處理時間上結構化索引遠比本文方法佔優勢。

綜合以上之效益評估，我們可以發現雖然本論文之索引方法比較耗時且在查準率上面表現較差，但卻能提供給使用者較多資訊，不但能找出符合檢索關鍵詞

**的資訊，也能找到語意上相關的資訊。雖然會提供給使用者較多的資訊，但卻能有效地增加檢索的查全率！**

# 第五章　結論與未來研究方向

本章總結本論文以及說明未來的研究方向，第一節說明本論文提出綜合兩種索引的方法在實作上的優點及其限制，第二節則說明本論文未來可能的研究發展方向。

## 第一節　結論

本篇論文提出一個結合結構化文件索引與語意索引的資訊檢索系統，提供一個檢索介面供使用者輸入關鍵詞，並藉由關鍵詞相關的語意找出有用的資訊回傳給使用者。

本文除了應用結構化文件索引，並引入語意索引的概念，利用[Chung99]的語意檢索技術，將「單純的詞鍵」轉換為「含有語意的概念空間」。並將結構化文件的結構資訊納入建置語意索引時的考慮，使得結構化文件的特性得以與語意索引合而為一，將傳統的資訊擷取提升至知識擷取的層次。

本篇論文所提出的方法具有下列特性：

1. 以「語意矩陣」及「概念空間」的觀念代替傳統的關鍵詞檢索，將使用者輸入的關鍵詞轉換成語意，更容易找出使用者感興趣的資料。
2. 將結構化文件的結構資訊納入語意索引的建置，考慮節點在階層式結構中的重要程度並賦予不同的權重，讓語意索引的建置可更深入文件內部的結構化資訊，而不僅止於文件內容。
3. 本文針對中文結構化文件作處理，著重在中文文件之知識擷取。
4. 相對於傳統計算權重的方式，本文考慮詞鍵於元素中出現頻率的索引方法，不僅能提供文件的元素層(Element Level)權重計算，也能動態且快速地還原文件本身(Document Level)的權重計算且僅需額外花費極少的計算時間。

## 第二節　未來研究方向

本文提出的方法是針對中文結構化文件來作語意索引，語意索引雖然可以利用「語意矩陣」及「概念空間」的觀念代替傳統的關鍵詞檢索，但在實作方面尚有些限制。從本文的效益評估中可發現，語意索引相當耗費計算時間，當文件總數倍增，則建置時間成指數成長，若欲應用在大量文件資料中，則必須配合平行計算的方法才有可能實現。然在實作的過程中，我們發現可藉由控制詞鍵數目來減少計算時間。欲控制詞鍵數目則必須在中文斷詞切字方面作更深入的研究，若能更精確地分析出每一個「真正有意義」的詞鍵，如此即使詞鍵個數減少也不會影響檢索的查準率。故未來的研究需朝如何更精確地對中文進行斷詞切字的處理邁進。

此外未來的研究方向亦可在檢索機制中加入相關性回饋(Relevant Feedback)的功能，利用使用者勾選檢索結果的正確性以自動修正詞鍵間的語意相似性，以期更有效地找出使用者真正想要的資料。

# 參考文獻


1. [Bourret00] R. Bourret, C. Bomhovd, and A. Buchmann; "A Generic Load/Extract Utility for Data Transfer Between XML Documents and Relational Databases," *Advanced Issues of E-Commerce and Web-Based Information Systems, 2000. WECWIS 2000. Second International Workshop,* 2000, Pages 134-143.
2. [Chow99] Jyh-Herng Chow, Josephine Cheng, Daniel Chang, and Jane Xu; "Index Design for Structured Documents Based on Abstraction," *6th International Conference on Database Systems for Advanced Applications*, 1999, Pages 89-96.
3. [Chung99] Ming Chung, Qin He, Kevin Powell and Bruce Schatz; "Semantic Indexing for a Complete Subject Discipline," *Proceedings of the fourth ACM conference on Digital libraries* , 1999, Pages 39 – 48.
4. [Dao98] Tuong Dao; "An Indexing Model for Structured Documents to Support Queries on Content, Structure and Attributes," *IEEE International Forum on Research and Technology Advances in Digital Libraries, ADL '98*, 1998, Pages 39 – 48.
5. [Frakes92] William B. Frakes, Ricardo Baeza-Yates, "Information Retrieval, Data Structures & Algorithms" 1992
6. [Grossman98] David A. Grossman, Ophir Frieder, "Information Retrieval: Algorithms and Heuristics" 1998.
7. [Han99] Sung-Geun Han, Jeong-Han Son, Jae-Woo Chang, and Zong-Cheol Zhoo; "Design and Implementation of a Structured Information Retrieval System for SGML Documents," *Database Systems for Advanced Applications, 1999.*

*Proceedings., 6th International Conference,* 1999, Pages 81-88.

8. [Kowalski97] Gerald Kowalski, "Information Retrieval Systems: Theory and Implementation," *Kluwer Academic Publishers*, 1997.
9. [Kasukawa99] Takeya Kasukawa, Hideo Matsuda, Michio Nakanishi, and Akihiro Hashimoto; "A New Method for Maintaining Semi-Structured Data Described in XML," *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, 1999, Pages: 258 -261.
10. [Lee96] Yong Kyu Lee, Seong-Joon Yoo, and Kyoungro Yoon; "Index Structures for Structured Documents," *Proceedings of the 1st ACM international conference on Digital libraries*, 1996, Pages 91-99.
11. [Myaeng98] Sung Hyon Myaeng, Don-Hyun Jang, Mun-Seok Kim and Zong-Cheol Zhoo; "A Flexible Model for Retrieval of SGML Documents," *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, 1998, Pages 138-145.
12. [Poullet97] Line Poullet, Jean-Marie Pinon, and Sylvie Calabretto; "Semantic Structuring of documents," *Proceedings of the Third Basque International Workshop on Information Technology*, 1997, Pages: 118 -124.
13. [Salton89] Gerard Salton, "Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer," *Addison-wesley publishing company*, 1989.
14. [Shin98] Dongwook Shin, Hyuncheol Jang, and Honglan Jin; "BUS: An Effective Indexing and Retrieval Scheme in Structured Documents," *Proceedings of the third ACM Conference on Digital libraries*, 1998, Pages 235 - 243.
15. [Wilkinson94] Ross Wilkinson; "Effective Retrieval of Structured Documents," *Proceedings of the 17th ACM SIGIR conference*, 1994, Pages: 311-317.

16. [Wolff00] Jens E. Wolff, Holger Florke, and Armin B. Cremers; “Searching and Browsing Collections of Structural Information,” *IEEE Proceedings of Advances in Digital Libraries*, 2000, Pages: 141-150.
17. **簡立峰**; “**中英文全文檢索技術及簡介**,” **國立成功大學圖書館館訊**, 82, **頁** 1-12.
18. **陳偉星**; “**應用短字串索引在中英文全文資料檢索之研究**,” **大葉學報**, 81, **頁** 161-173.
19. **王良志**, **貝子勝**, **黎偉權**, **黃麗傾**; “**以剖析為導向的中文斷詞法**,“ **電腦軟體技術專刊**, 80, **頁** 40-45.