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

針對圖形資料庫搜尋前 K 名 R-cliques

Finding Top-K R-cliques in a Graph Database

研究生：張餘恩撰

中華民國 103 年 7 月

針對圖形資料庫搜尋前 K 名 R-cliques

Finding Top-K R-cliques in a Graph Database

研究生：張餘恩　Student：Yu-EnChang

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 Scienece
National Taiwan Ocean University
In Partial Fulfillment of the Requirements
for the Degree of
Master of Science
in
Computer Science and Engineering
July2014
Keelung, Taiwan, Republic of China

中華民國 103 年 7 月

摘要

圖型資料庫(Graph database)能夠明顯表現出資料之間的相關性，而關鍵字搜尋提供使用者便利的查詢工具，所以如何在圖型資料中進行關鍵字搜尋成為一個重要的議題。先前的研究提出以 *r*-clique 作為關鍵字搜尋圖型資料庫回傳的結果，其為一群擁有所有查詢關鍵字且對應節點之間的距離小於一定值的節點叢集。雖然 *r*-clique 的定義可以有效地尋找出有意義的資料，但是之前所提出用以找出所有解的 Branch and Bound 演算法，和找出 top-*k* 名近似解的 Polynomial Delay 演算法，其所需時間皆相當龐大。本論文利用將圖形資料轉換為樹狀的方法，並提出了以搜尋 ELCA 為基礎的演算法，以在樹狀結構中搜尋出前 *k* 名的 *r*-clique。實驗結果顯示，我們提出的方法在出現頻率高的關鍵字查詢中，時間效率皆優於先前研究所提出的演算法。

關鍵詞：圖形資料庫、關鍵字查詢、*r*-clique。

Abstract

Graph databases have been applied to represent complex data, and keyword search on graph databases is a convenient mechanism for users. Therefore, it is an important issue to improve the efficiency on searching graph databases. The earlier research proposed the idea of“*r*-clique” as the results of keyword search on graph databases, where an *r*-clique contains every query keyword and the distance among every keyword match is under a given threshold *r*. The earlier research has proposed the Branch and Bound algorithmto find all *r*-cliques, and the Polynomial-Delay algorithm to find approximatetop-*k* answers, but they both require a lot of computation costs. In this thesis, we study how to convert a graph database to a tree database, and use tree search methods to generate top-*kr*-cliques in a graph database.We propose theSRE algorithm. It is a search method based on ELCA to get search results efficiently. We also propose three strategies to convert a graph database into a tree database.Finally, we design a series of experiments to compare our approach with existing approaches. Experimental results show that when the input keyword has high frequency, our method performs more efficiently than existing approaches.

Keyword: graph database, keyword search, r-clique

誌謝

首先,感謝指導教授張雅惠博士,對於本論文給予相當鼎力的協助,且在論文研究期間不時地共同討論，以及悉心指正與釋疑，並彌補學生於專業學識上的不足。由於老師的耐心督導，使本論文得以順利地完成，同時也讓學生獲益匪淺，在此向敬愛的老師致上衷心的謝忱。

除此之外，感謝口試審查委員劉傳銘博士與許為元博士，細心審稿以及論文修正，並對於本論文不吝提供寶貴的建議，使本論文更趨於嚴謹完善，在此深表謝意。

感謝我的大學同班同學陳柏諺，在課餘總是不厭其煩與我討論，在論文完成過程中給予我許多幫助。感謝實驗室的學長姊同學與學弟妹與其他實驗室的同學們，陪伴我渡過多采多姿的實驗室生活。最後，感謝親愛的家人，於學習成長階段給予許多支持與鼓勵。在此一併致上心中無限的感激，謝謝你們。

目錄

圖目錄

表目錄

第 1 章序論

本章節說明本論文的研究動機、目的和研究的方法，以及提出本論文的貢獻，並介紹相關的研究，最後說明本論文的架構以及各章節的內容。

1.1 研究動機與目的

近年來針對在圖形化資料進行關鍵字搜尋的研究越來越多，而圖形化資料不僅能以資料本身的資訊來表達意義，更能靠著資料與資料間的關係來呈現如相關度之類本身資料所沒有含有的資訊。此外，許多傳統資料庫在經過特定的規則後也能轉化成圖型資料庫，如關連式資料庫能憑藉資料的 foreign key 建立與其他表格資料之間的關係，進而轉換成圖形資料。

許多人提倡以簡易的關鍵字查詢對圖型資料進行查詢，而[KA11]提出了 *r*-clique 的搜尋方式對圖型資料進行關鍵字查詢。在一個 *r*-clique 中，針對所有關鍵字至少包含一個對應的節點，且所有對應節點之間的距離皆小於一個限制值 R。然而在圖型資料庫能夠呈現資料關係的背後，同時也必須處理因圖型資料庫的結構太過複雜而使查詢效率較慢的問題。傳統資料庫如關連式資料庫和樹狀資料庫因為結構較單純，加上近年來已經有相當多的研究提出了許多成熟的改良方法，在搜尋的效率上都已經有相當的改善。但是圖型資料庫在結構先天上處理的時間複雜度已經比傳統資料庫的結構來的複雜許多，若再考慮資料的大小較大的問題，圖型資料庫的搜尋效率將會遠低於傳統資料庫的搜尋。

[KA11]提出 Branch and Bound 演算法來搜尋一個圖所含有的全部 *r*-clique。此演算法流程如下：一開始使用者必須給定一個限制值 *r*，之後 Branch and Bound 演算法會從欲搜尋的圖中取出所有符合搜尋關鍵字的節點，並依照關鍵字對每個節點作分群。接著從第一個關鍵字的節點群作為第一群，對第二個關鍵字的節點群逐一檢查，若有第一群的節點和第二群的某個節點距離小於 R，則將這兩個節

點結合成一個節點集合,作為後補叢集放在一個後補叢集集合(Candidate set)內，在這兩群的所有節點檢查完之後,再將候補叢集集合中所有的後補叢集與第三個關鍵字的節點群一一比較節點距離,若後補叢集所有的節點和第三群的某個節點距離皆小於 r，則將此後補叢集和第三群中被檢查的節點形成新的後補叢集並插入到新的後補叢集集合中。Branch and Bound 演算法會不斷進行這個檢查動作直到所有的關鍵字集合皆被檢查過,最後留下的後補叢集集合中便是符合查詢所要回傳的 r-clique 集合。但是 Branch and Bound 演算法的時間複雜度過高，若假設查詢句有 m 個關鍵字且$|C_{MAX}|$為擁有最多對應節點的集合， Branch and Bound 演算法的時間複雜度則為 $O(m^2 |C_{MAX}|^{m+1})$。因此，[KA11]隨之提出了 Polynomail-Delay 演算法以快速輸出 top-k 名 r-clique，此演算法可將搜尋範圍分割為多個搜尋子空間，使得計算量減少，假設查詢句有 m 個關鍵字且$|S_{MAX}|$為擁有最多對應節點的搜尋(子)空間大小，而將搜尋複雜度改善為 $O(m^2 |S_{MAX}|^2)$，但因為沒有搜尋所有可能的狀況，其輸出僅為近似解。

於目前的做法還有改善的空間,[張 13]提出了透過將圖形資料轉換為樹狀的結構，並在樹狀的結構中以搜尋 ELCA (Exclusive Lowest Common Ancestor) [ZBWL+12]節點的方法來搜尋所有 r-clique,以有效降低了搜尋的範圍與複雜度。而本論文則是進一步探討,此做法在輸出 top-k 的情況下,其有效性與效率如何。

1.2 研究方法與貢獻

在樹狀資料中，使用關鍵字搜尋時，我們可以找出各個包含所有對應節點的子樹來表示各個對應節點之間的關係，而針對關鍵字搜尋我們可以使用 LCA (Lowest Common Ancestor)搜尋來找出一組對應節點的組合所形成的子樹，該子樹的根節點便為該組關鍵字組合的 LCA 節點。但是 LCA 搜尋所找到的結果太多，需要進一步的篩選較有意義的 LCA 節點。在本論文中，我們採用 LCA 的變形，

亦即 ELCA (Exclusive Lowest Common Ancestor) [ZBWL+12]，來找出一組組包含對應節點的子樹。但 ELCA 節點的形成也受樹狀結構影響，如此會導致能成為 *r*-clique 得節點會被排除在搜尋範圍之外，所以我們將圖形資料轉換為多種樹結構，進行實驗觀察與比較，在何種樹狀結構中，效率和回傳解較佳。

本論文以[張13]所提出的樹狀搜尋方式為基礎，針對以樹狀結構搜尋 *r*-clique 結果，做 top-*k* 排序。並透過實驗，驗證我們方法的有效性與效率。

針對本論文主要貢獻，總結如下所示：

1. 我們利用[張 13]提出的樹狀轉換策略，降低在圖形資料中搜尋 *r*-clique 的複雜度。

2. 我們提出 Search R-clique based on ELCA(SRE)演算法，針對透過對樹狀資料搜尋 *r*-clique 的結果，並以樹上邊的 *weight* 總和排序輸出 top-*k* 名 *r*-clique。

3. 我們透過實驗，應證了所提出的方法依然能達到與 Branch and Bound 近似的解，且在時間效率上較 Polynomial-Delay 演算法佳。

1.3 相關研究

首先我們討論針對資料圖進行關鍵字查詢的相關研究。[DLQW+07]提出了對關連式資料庫的資料轉換成資料圖的方法，並從其中找出 top-*k* 的 Steiner tree。研究[LOF+08]提出稱作 EASE 的方法，以便對 Unstructured、Semi-structured 和 Structured Data，皆可做關鍵字的查詢。EASE 的作法也是將 data 表示成 graph，然後利用 adjacency matrix，解決 *r*-radius Steiner graph 的問題。研究[TJM+08]提出讓使用者在不需要知道複雜的 schema 以及 query language 的情況下，能建構出查詢句的 template。該論文首先將 keyword match 到 schema graph 上，並附上每個資料來源的權重值，然後在 schema graph 上找到 top-*k* 的 Steiner tree，以產生 top-*k* 的 query。該系統會進一步針對所產生的查詢句提供答案以及對應的資料

來源，以便使用者提供 feedback 給系統，讓系統可以學習並改變權重值。研究[YCHH12]則探討針對結構化資料關鍵字查詢改寫的問題，這些改寫後的查詢提供了代替原輸入的描述，進而可以更好的捕捉使用者的資訊需求。改寫過程分為在 off-line 階段產生 Term Augmented Tuple Graph，並加入一個新的 preference vector 參數以抓取具有相關性的 term，到 on-line 階段時使用一有效的機率性產生模組，來產生改寫的查詢句。至於[IBS08]則整理出泛用的 Top-*k* 演算法。

和本論文所探討的問題最為近似的是研究[QYCT09]和[KA11]。[QYCT09]提出了對關連式資料庫搜尋 community 的作法，所謂的 community 有一個中心點，然後每一個對應節點到中心點的距離皆小於一個給定的值。為了改善 Graph database search 搜尋結果的召回率以及資料的可讀性，[KA11]進一步提出 *r*-clique 的定義作為查詢的結果，並針對 *r*-clique 的搜尋提出兩種演算法。第一種是利用 Branch and Bound 演算法一一檢視可能形成答案的節點組合以取得所有 *r*-clique，另一種是 Polynomial-Delay，利用分割搜尋空間來尋找近似 *r*-clique 的搜尋結果，同時也可以利用各個子搜尋空間的排序來支援 top-*k* 的搜尋。

至於特別針對 XML 文件進行關鍵字搜尋的部分，[XP05]提出 SLCA 的定義，受到廣泛的重視與引用。[PX07]則修正其定義而後提出 ELCA 的定義，以獲得一些 SLCA 無法取得的資訊。[ZBWL+12]針對關鍵字查詢的 SLCA 以及 ELCA 提出了比以往更快速的搜尋方法，其方式是分別對關鍵字產生相對應直接或間接擁有對應節點 ID 的 inverted list，基於 intersection operation 提出了全新完整有效的演算法，使得問題即是在 inverted lists 中尋找符合一定條件的節點，實驗結果顯示其方法優於其他研究的做法一至二個 order。論文[TPSS11]則提出對 XML 文件指定一節點，找出含有查詢關鍵字且最接近該節點的節點搜尋方式，更將此搜尋方式延伸到 XPath 上來改善 XPath query 的效率，同時提出能有效配合上述搜尋方式的索引格式。

1.4 論文架構

本論文其餘各章節的架構如下：第二章介紹本論文所會使用說明基本的定義，包括圖型資料的表示法、*r*-clique 的定義及何謂 top-*k* 結果。同時介紹其他研究針對相同問題的 Branch and Bound (為便於說明接下來在本論文中簡稱 Branch&Bound)演算法和 Polynomial-Delay 演算法。在第三章我們先介紹與資料樹搜尋相關的基本定義如 LCA、SLCA、ELCA，再介紹如何轉換圖型資料至樹狀資料，並說明我們所提出的 SRE 演算法，如何在樹狀結構上以搜尋 ELCA 的方式回傳 *r*-clique 並輸出 top-*k* 組結果。在第四章實驗中，以所提出的三個樹狀結構透過 SRE 演算法搜尋的方法與 Branch&Bound、Polynomial-Delay 的方法來比較效率和查詢有效性。最後在第五章提出結論以及本論文未來的研究方向。

第 2 章基本定義和相關做法

我們在此章說明基本的定義，包括圖型資料的表示法、*r*-clique 的定義及何謂 top-*k* 結果。接著，介紹其他研究針對相同問題的 Branch&Bound 演算法和 Polynomial-Delay 演算法。

2.1 圖形資料 (data graph) 表示法

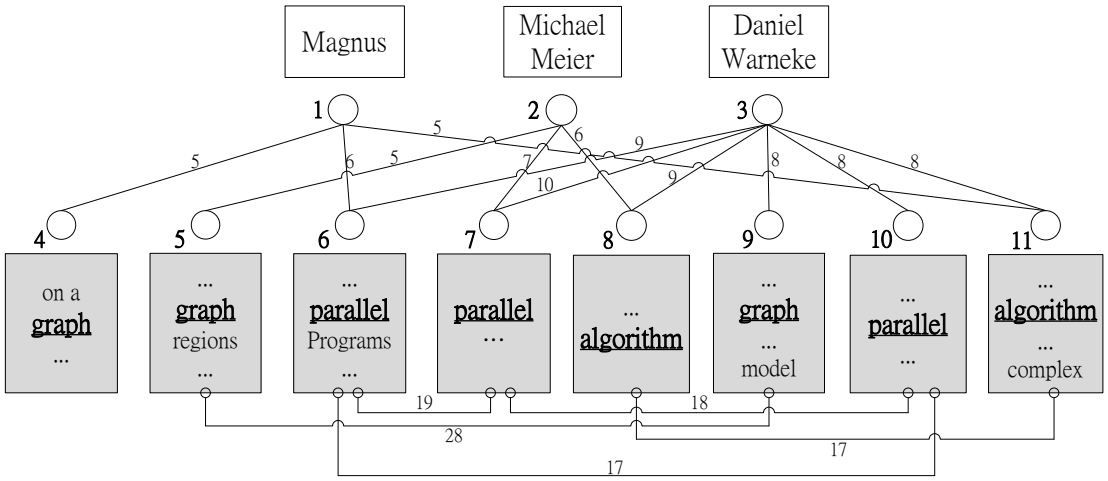


圖 2.1 範例圖形資料

首先介紹本論文處理的圖形資料。一個資料圖中含有多個節點，每個節點被指定一個不會重複的編號，且各自擁有的關鍵字資訊。如圖 2.1 中，節點 1-3 代表作者，節點 4-11 代表論文。節點中間可以利用線段代表存在特定關係，例如作者 1 號撰寫論文 4 號、6 號和 11 號，而論文間則存在引用的關係，每個線段可以利用不同的公式計算兩端點的相關性，稱作距離或 *weight*，在此圖 2.1 中的 *weight* 是因應舉例所需特意指定，而沒有套用任何公式，注意，在此 *weight* 值越小表示相關性越高。接下來的章節，我們先提出定義以方便後續的說明。

2.2 關鍵字查詢與輸出 top-*k* 結果

在此節中，我們依照研究[KA11]的定義，將本論文會使用到的定義說明如下：

[定義 2.1]：對應節點(match) ：給定一個資料圖 *G* 或樹 *T* 以及查詢句 *Q*，在圖 *G* 或樹 *T* 中的節點 *n* 若含有能對應查詢句 *Q* 的關鍵字，我們稱 *n* 為 *Q* 對圖 *G* 或樹 *T* 中的對應節點(match)。

[定義 2.2]：*r*-clique ：給定一個資料圖 *G* 及限定距離大小值 threshold *r*，查詢句 $Q = \{k_1, k_2, \ldots, k_l\}$，一個 *r*-clique 代表在 *G* 中的一組節點集合，針對查詢句 *Q* 中的所有關鍵字皆有對應節點，且每一節點至 *r*-clique 中另一個節點的距離皆小於 *r*。在 *r*-clique 中的節點距離為，在圖 *G* 中兩個節點的最短距離。

[範例 2.1] ：在圖 2.1 中，我們希望找出與 parallel、graph、algorithm 這三個關鍵字有一定相關程度的論文，對查詢句關鍵字下“parallel”, “graph”, “algorithm”，*r* 值定為 20 ，八組 *r*-clique 分別為{4, 6, 11}、{5, 7, 8}、{6, 8, 9}、{6, 9, 11}、{7, 8, 9}、{7, 9, 11}、{8, 9, 10}、{9, 10, 11}，如圖 2.2，在該圖中兩個節點的距離皆為圖 2.1 中兩節點所求得的最短路徑距離。

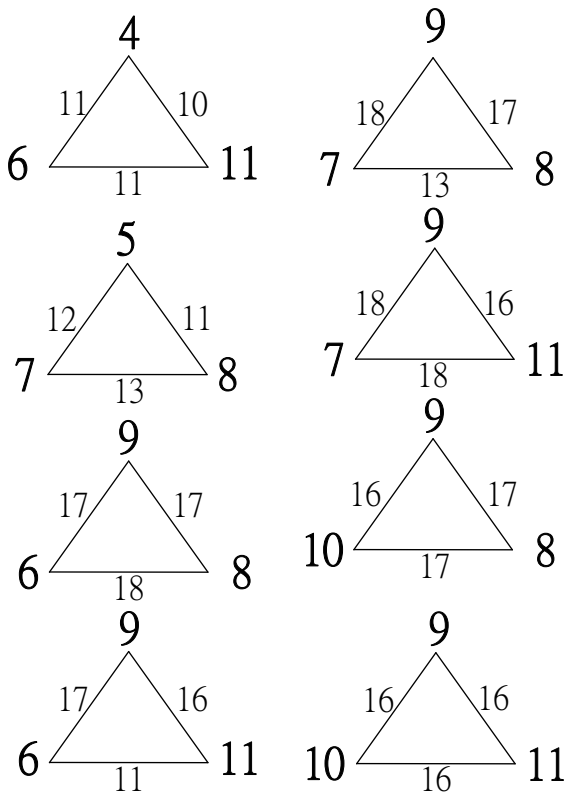


圖 2.2 範例 *r*-clique

[定義 2.3]：MinimalSteiner Tree (MST) ：給定一個資料庫 *G* 及一個 *r*-clique *C*，在圖 *G* 中一個包含所有 *C* 的節點的 sub-tree 稱作 Steiner tree，而其中 *weight* 總和最小的稱為 Minimal Steiner tree，簡稱 MST。

[範例 2.2]：針對一個 *r*-clique，在原圖中可找到一個邊距離總和最小且包含該 *r*-clique 的所有關鍵字節點的 MST。例如圖 2.2，*r*-clique{4, 6, 11}對應回圖 2.1 可得到圖 2.3 中的 MST {1, 4, 6, 11}，其 *weight* 總和 5+6+5=16。

[定義 2.4]：top-*k* 結果：將所有符合的 *r*-clique 在 MST 中實際邊距離(*weight* 值)總和，由小到大排序，並輸出前 *k* 名，即為本論文所要求之 top-*k* 結果。

[範例 2.3]：參考範例 2.1，八組 *r*-clique 分別為：{4, 6, 11}、{5, 7, 8}、{6, 8, 9}、

{6, 9, 11}、{7, 8, 9}、{7, 9, 11}、{8, 9, 10}、{9, 10, 11}，且其各自對應的 MST 與 *weight* 總和，如圖 2.3，此時查詢 top-*k* 之 *k* 值設定為 5 由小到大排序後，如圖 2.4，輸出前 5 個 *r*-clique：{4, 6, 11}、{5, 7, 8}、{7, 8, 9}、{9, 10, 11}、{6, 9, 11}。

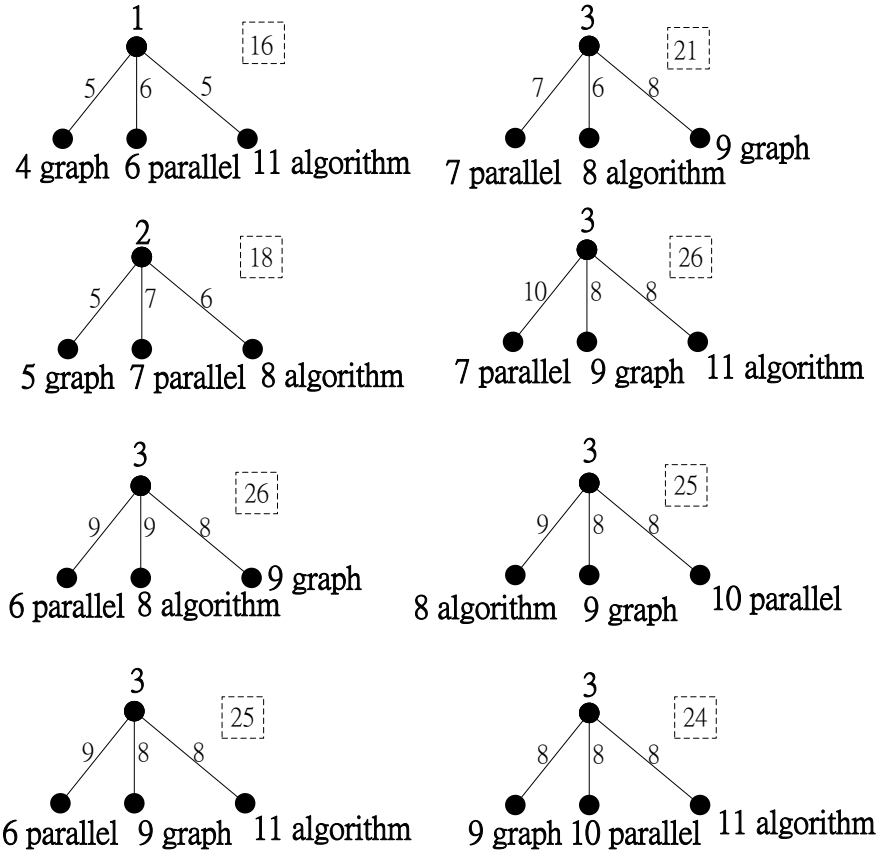


圖 2.3 範例 MinimalSteiner Tree (MST)

top-*k* List	
r-clique	*weight*
4,6,11	16
5,7,8	18
7,8,9	21
9,10,11	24
6,9,11	25
8,9,10	25
6,8,9	26
7,9,11	26

圖 2.4 範例 top-*k* List

2.3 基本做法(Baseline approach)

[KA11]針對 *r*-clique 的定義，提出了 Branch&Bound 演算法以找出圖中所有的 *r*-clique，其做法是，將圖中所有符合關鍵字的節點會依照關鍵字分群，之後再一一檢查距離是否小於 *r* 能否形成 *r*-clique，在找出所有的 *r*-clique 之後再針對各個 *r*-clique 呼叫 DrawSteinerTree 演算法將每個 *r*-clique 對應的 Steiner tree，最後利用排序演算法找出 top-*kr*-cliques。以下說明 Baseline 演算法：

演算法名稱：Baseline Algorithm

輸入：資料圖 G, 查詢句$\{k_1, k_2, ..., k_m\}$共 m 個關鍵字, 距離限制 r, 以及 top-k 的 k 值

輸出：所有符合搜尋的 top-k 名 r-clique

```
L01:  rList ← Branch&Bound(G, {k1 , k2 , ... , km}, r)
L02:  foreach RC in rList do
L03:        SteinerTreeSet.add(DrawSteinerTree(RC, G))
L04:  Top-kList←rank(SteinerTreeSet, r )
L05:  return Top-kList
```

圖 2.5 Baseline 演算法[KA11]

如圖 2.5 主要步驟分成三個階段，第一個階段是在 L01 呼叫 Branch&Bound 演算法得到所有的 r-clique 並存入到 rList 中，第二個階段是在 L02~L03 將 rList 中所有的 r-clique 依照原始資料圖 G 畫成 Steiner tree，最後在 L04~L05 排序並回傳 top-k 名 r-clique。以下我們進一步解釋 Branch&Bound 演算法與 DrawSteinerTree 演算法。

演算法名稱：Branch&Bound

輸入：資料圖 G, 查詢句$\{k_1, k_2, \ldots, k_m\}$共 m 個關鍵字, 距離限制 r

輸出：所有符合搜尋的 r-clique

```
L01:  for i← 1 to m do
L02:      C_i← the set of nodes in G containing k_i
L03:  rList ← empty
L04:  for i← 1 to size( C_1 ) do
L05:      rList.add(C_i^1 )
L06:  for i← 2 to m do
L07:      newRList ← empty
L08:      for j← 1 to size( C_i ) do
L09:          for k← 1 to size(rList) do
L10: if ∀ node ∈ rList_k dist(node,C_i^j ) ≤ r (whererList_k is the kth
  element of rList) then
L11:                  newCandidate ←C_i^j .concatenate(node)
L12: newRList_k .add(newCandidate)
L13:      rList ← newRList
L14:  return rList
```

圖 2.6Branch&Bound 演算法

Branch&Bound 如圖 2.6 所示。首先，在 L01 和 L02 將每個節點依照 m 個關鍵字分成 m 群，在 L03 建立 rList 作為存放後補叢集的集合，接著在 L04 和 L05 將第一個關鍵字的節點群存入 rList 後，在 L06 開始進入迴圈，在 L07 建立空的 newRList 用來存放接下來運算結束所得到的新的後補叢集的集合。第一次進入迴圈後會將 rList 中的所有節點和第二個關鍵字的所有節點比較距離，一但距離小於 r，便將檢查到的 rList 中所有的節點和第二群中檢查到距離小於 R 的節點組合成新的後補叢集，再將這個後補叢集放入 newRList 中，再將第二群的所有節點檢查完之後，把 rList 的內容取代成 newRList 所存放的後補叢集集合。再進

行下一次的迴圈，在迴圈中檢查完所有的關鍵字集合後得將所求得的 *r*-clique 存入 rList，在迴圈結束後從最後回傳 rList 取得所有 *r*-clique。

演算法名稱：DrawSteinerTree 輸入：圖型資料庫 *G*, 一個 *r*-clique RC 輸出：*r*-clique RC 的 steiner tree
L01: Let G_1 be *r*-clique RC. L02: Find the minimal spanning tree T_1 of G_1 L03: Create graph G_2 by replacing each edge in T_1 by itscorresponding shortest path in *G*. L04: Find the minimal spanning tree T_2 of G_2. L05: Create a Steiner tree from T_2 by removing the leaves (and the associated edges) that are not in the *r*-clique. L06: return T_2

圖 2.7 DrawSteinerTree 演算法

為了顯示每個 R-clique 中的點在原圖中的關係，[KA11]針對各個 *r*-clique 對應圖 *G* 呼叫 DrawSteinerTree 演算法畫出對應的 Steiner tree。找出 Minimal Steiner Tree 基本上是個 NP 的問題，如圖 2.7 所示的為一個找近似解的方法。其步驟主要分成下列的幾個部分，首先必須先對輸入的 *r*-clique RC 畫出 Minimal spanning tree T_1，之後將 T_1 的所有邊替換成邊上的兩個點在圖 *G* 上的最短路徑，建構出圖 G_2，再對圖 G_2 畫出 G_2 的 Minimal spanning tree T_2，最後再針對 T_2 將所有葉節點不屬於 RC 的點從 T_2 中剔除，直到所有葉節點皆為 RC 的點為止最後回傳 T_2 即為 RC 的 Steiner tree。

此 Baseline 方法雖然可以找出有效的答案，但計算量很大，以下我們討論其時間複雜度。針對 Branch&Bound，假設所有節點的距離皆小於 r 且$|C_{MAX}|$為擁有

最多對應節點的集合，由於在圖 2.6 中 L06 需要作 m-1 次，L08 在最差的狀況下必須作$|C_{MAX}|$次，而 L09 到 L12 部分 rList 內的數量會不斷增加，最後會變成有$|C_1|*|C_2|*\cdots*|C_m|$個，在最差的狀況會達到$|C_{MAX}|^m$個。另外在 L10 檢查 dist 的部分由於有 m 個關鍵字故最多必須檢查 m 次，故 Branch&Bound 演算法的時間複雜度為 O(m^2 $|C_{MAX}{}^{m+1}|$)。DrawSteinerTree 演算法的部分我們假設 RC 的點為 S，圖 G_1 的點為 V，則可以在 O($|S||V|^2$)的時間建出 G_1，而畫出 Minimal spanning tree T_1 可以在 O($|S|^2$)完成，而建構 G_2 可以在 O($|V|$)的時間完成，畫出 Minimal spanning tree T_2 只需要 O($|V|^2$)，最後剔除非 r-clique 節點的步驟可以在 O($|V|$)完成，故 DrawSteinerTree 演算法的時間複雜度為 O($|S||V|^2$)。

以下我們用範例圖 2.1 來說明 Branch&Bound 演算法從頭到尾，如何搜尋輸出 top-k 名 r-clique 以及 Minimal Steiner Tree：

假設我們下達的查詢句為：關鍵字“graph”,“parallel”,“algorithm”，r= 20，k = 5。Branch&Bound 演算法首先會先將圖 2.1 上的對應節點依照關鍵字分群，如圖 2.8：

關鍵字	節點		
graph	4	5	9
parallel	6	7	10
algorithm	8	11	

圖 2.8 節點分群

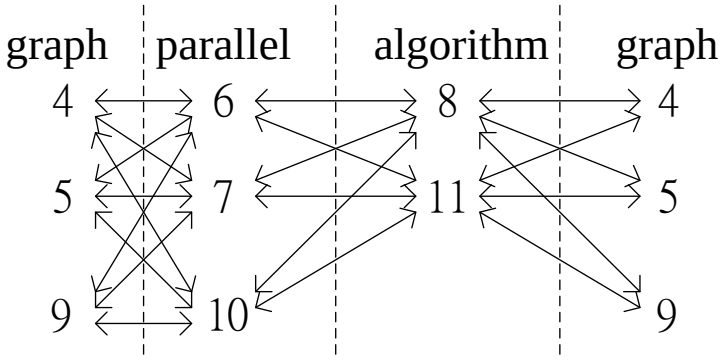


圖 2.9 計算兩兩節點之最短距離

接下來計算兩兩節點之間在圖 2.1 中的最短距離，如圖 2.9，每一條實線箭頭就代表要計算一次最短距離是否小於 *r* 值。並且判斷是否形成 *r*-clique ，如圖 2.10。

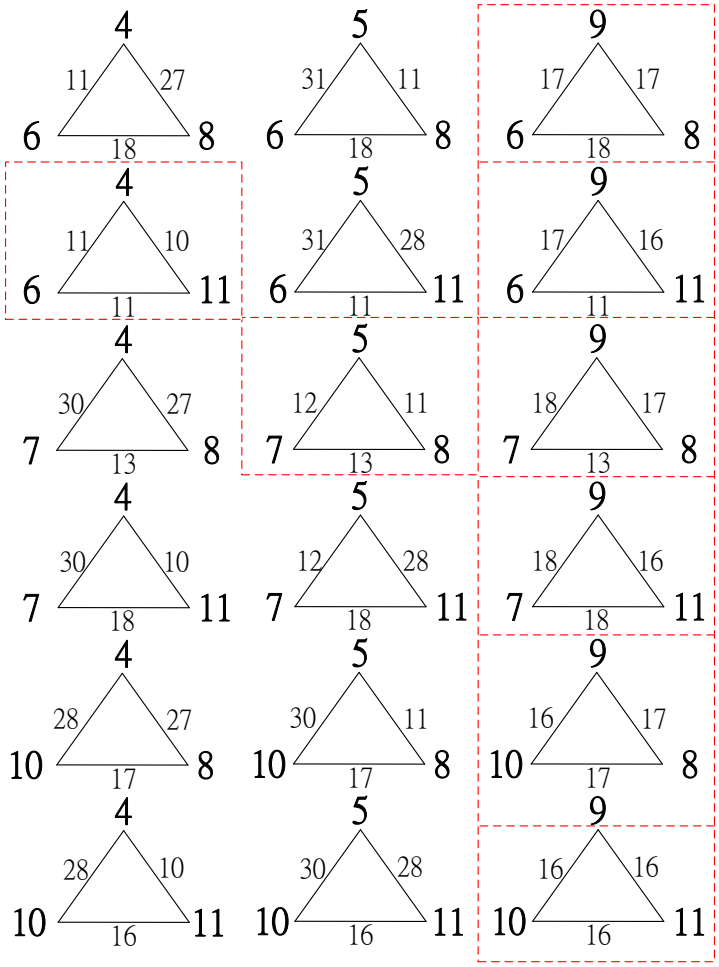


圖 2.10 判斷是否形成 *r*-clique

如圖 2.10，虛線框起的 clique 組合即是符合查詢句限制的 *r*-clique，符合查詢句關鍵字及 *r* 值的條件 *r*-clique 組合有八組，分別為{4, 6, 11}、{5, 7, 8}、{6, 8, 9}、{6, 9, 11}、{7, 8, 9}、{7, 9, 11}、{8, 9, 10}、{9, 10, 11}，並將所有 *r*-clique 回傳，對應如前面圖 2.2。

接著，由圖 2.1 中搜尋並回傳所有 *r*-clique 的 MST(Minimal Steiner Tree)，如前面圖 2.3。以節點 1 為跟節點的 MST：{1, 4, 6, 11}，以節點 2 為跟節點的 MST：{2, 5, 7, 8}，以節點 3 為跟節點的 MST：{3, 6, 8, 9}、{3, 6, 9, 11}、{3, 7, 8, 9}、{3, 7, 9, 11}、{3, 8, 9, 10}、{3, 9, 10, 11}。

最後，如前面圖 2.4 依照 MST 的 *weight*(即 MST 所有邊之總和)，排序輸出 top-*k* 名，由於查詢句 *k* 值設定為 5，所以這裡輸出前 5 名 *r*-clique：{4, 6, 11}、{5, 7, 8}、{7, 8, 9}、{9, 10, 11}、{6, 9, 11}。

也因為如此複雜的計算 Branch&Bound 演算法的時間複雜度為 $O(m^2|C_{MAX}|^{m+1})$(m 為關鍵字個數，$|C_{MAX}|$為含有關鍵字之節點個數的最大值)，故[KA11]以輸出 top-*k* 名 *r*-clique 為目標，提出 Polynomial-Delay 演算法來改善，接下來我們繼續依圖 2.1 之範例來介紹 Polynomial-Delay 演算法。

2.4 Polynomial－Delay 演算法

為了提升效率，[KA11]另外提出一個 Polynomial-time 的演算法，以找出近似解，此演算法也是先將圖形資料中的節點，依照關鍵字分群，不過此方法會由節點一一搜尋此分群節點到其他分群為最短距離的節點，首先找到排名最佳的 *r*-clique 節點回傳，再透過原分群刪減最佳 *r*-clique 節點形成若干子分群來搜尋次佳 *r*-clique 節點回傳，依此類推。

以下列出 Polynomial-Delay 主要演算法，此演算法基本上取代圖 2.5 的 L01 和 L04 的做法：

演算法名稱：Polynomial-Delay 輸入：the input graph G; the query $\{k_1; k_2; ... ; k_l\}$; r and$k$ 輸出：the set of top-k ordered r-cliques printed withPolynomial Delay
L01: for i←1 to l do L02: C_i←the set of nodes in G containing k_i L03: C← 〈$C_1, C_2, . . . , C_l$〉 L04: Queue ← an empty priority queue L05: A← FindTopRankedAnswer(C, G, l, r) L06: if $A \neq \emptyset$ then L07: Queue.insert(〈A, C_i〉) L08: while Queue ≠∅do L09: 〈A, S〉← Queue.removeTop() L10: print(A) L11: k←k – 1 L12: if k = 0 then L13: return L14: for i← 1 to l do L15: S_i←S.get(i) L16: 〈$SB_1, SB_2, . . . , SB_l$〉← ProduceSubSpaces(〈$A,S_1, . . . , S_l$〉) L17: for i← 2 to l do L18: A_i← FindTopRankedAnswer(SB_i, G, l, r) L19: if $A_i \neq \emptyset$ then L20: Queue.insert(〈A_i, SB_i〉)

圖 2.11Polynomial-Delay 演算法

如圖 2.11，Polynomial-Delay 演算法，在 L01~L04，依照關鍵字分群後，L05 呼叫 FindTopRankedAnswer，以關鍵字分群下各節點為出發點，搜尋該節點到另外兩個關鍵字群為最短距離之節點，並將距離總和排序後回傳最小之節點組合即

為第一名的 *r*-clique 節點組合。L07 為回傳 top-*k* 名 *r*-clique 的 Queue，L08~L13 輸出 top-*k* 名的 *r*-clique 結果，L14~L18，呼叫 ProduceSubSpaces，將原本的搜尋空間依序排除最佳解節點，成為搜尋子空間，並將各個搜尋子空間找出最佳解，並排序在第一名的 *r*-clique 之後，L19~L20 為回傳所有搜尋子空間的最佳解。

以下我們用前面圖 2.1 來說明 Polynomial-Delay 演算法從頭到尾，如何搜尋輸出 top-*k* 名 *r*-clique，首先 Polynomial-Delay 演算法也會先將節點依照關鍵字分群，圖 2.8。

接著，將分群好的節點，以關鍵字分群下各節點為出發點，搜尋該節點到另外兩個關鍵字群為最短距離之節點，並將距離總和排序後回傳最小之節點組合即為第一名的 *r*-clique 節點組合，如下圖 2.12。圖 2.12(a)為以關鍵字“graph”為出發點，搜尋到另外兩關鍵字分群之節點最短距離組合，圖 2.12(b)為為以關鍵字“parallel”為出發點，搜尋到另外兩關鍵字分群之節點最短距離組合，圖 2.12(c)為為以關鍵字“algorithm”為出發點，搜尋到另外兩關鍵字分群之節點最短距離組合。

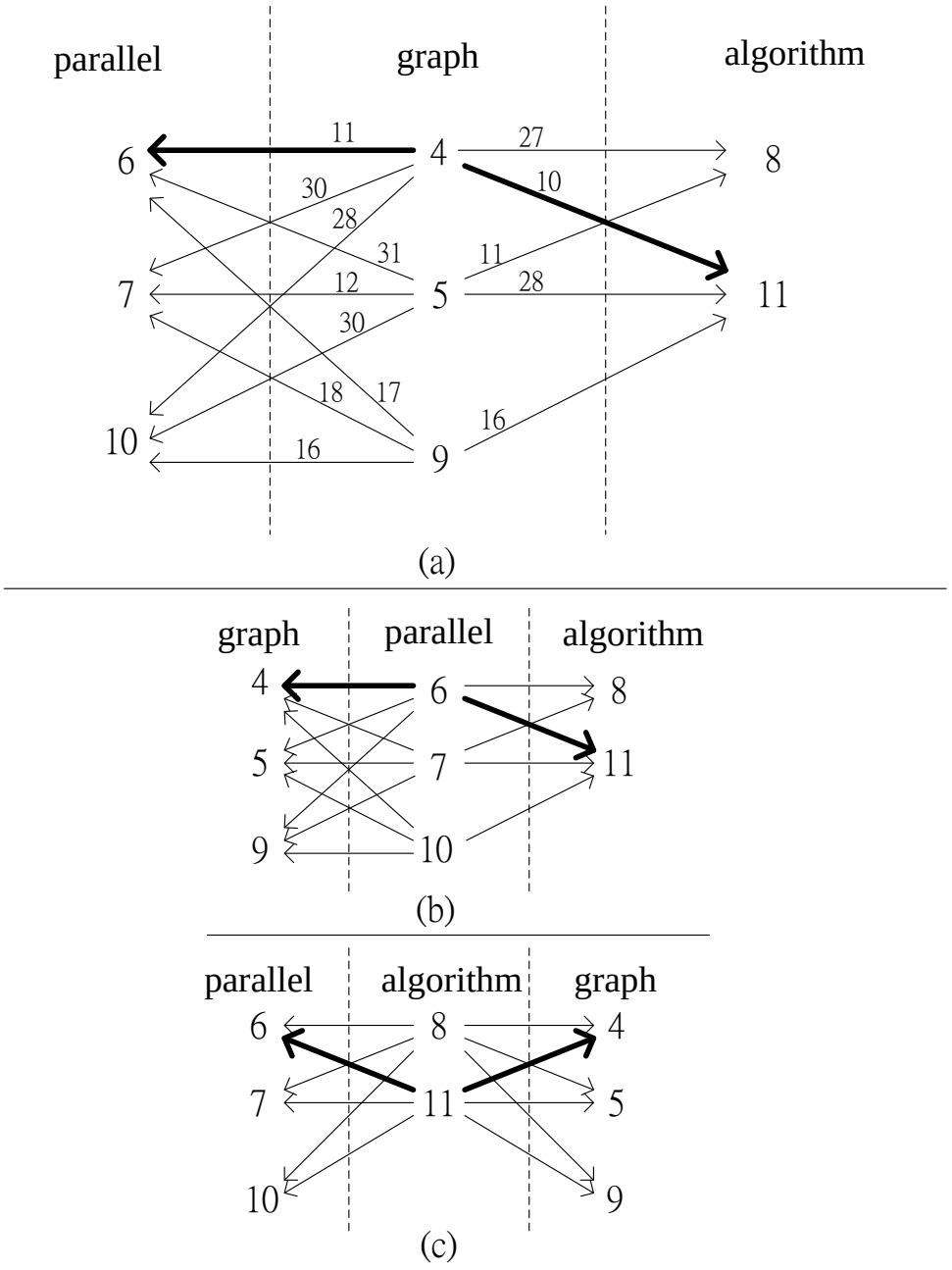


圖 2.12 搜尋各分群至另外兩分群之所有節點組合

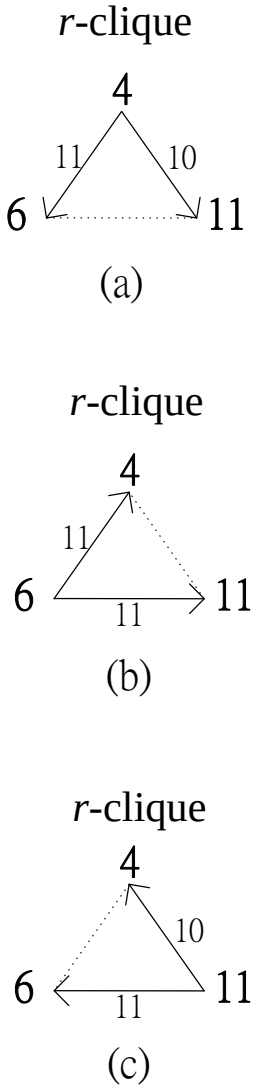


圖 2.13 最小 *weight* 值節點組合

我們以圖 2.12(a)為例說明，以關鍵字“graph”分群下節點 4、5、9 為出發點，可以找到在所有的組合中，以節點 4 到節點 6、節點 4 到節點 11，為圖 2.12(a)之最短距離節點組，並將{4, 6, 11}回傳，對應如圖 2.13(a)。圖 2.13 可以看到，以三種關鍵字群為出發點的找出來的最小 *weight* 值組合正好皆為{4, 6, 11}，故將{4, 6, 11}輸出為 top-1 的 *r*-clique。

現在假如我們要往下尋找次佳解，依照演算法將原本的搜尋空間依序排除最佳解節點，成為搜尋子空間，如圖 2.14：

關鍵字	節點		
graph	5	9	
parallel	6	7	10
algorithm	8	11	

(a)

關鍵字	節點		
graph	4		
parallel	7	10	
algorithm	8	11	

(b)

關鍵字	節點		
graph	4		
parallel	6		
algorithm	8		

(c)

圖 2.14 搜尋子空間

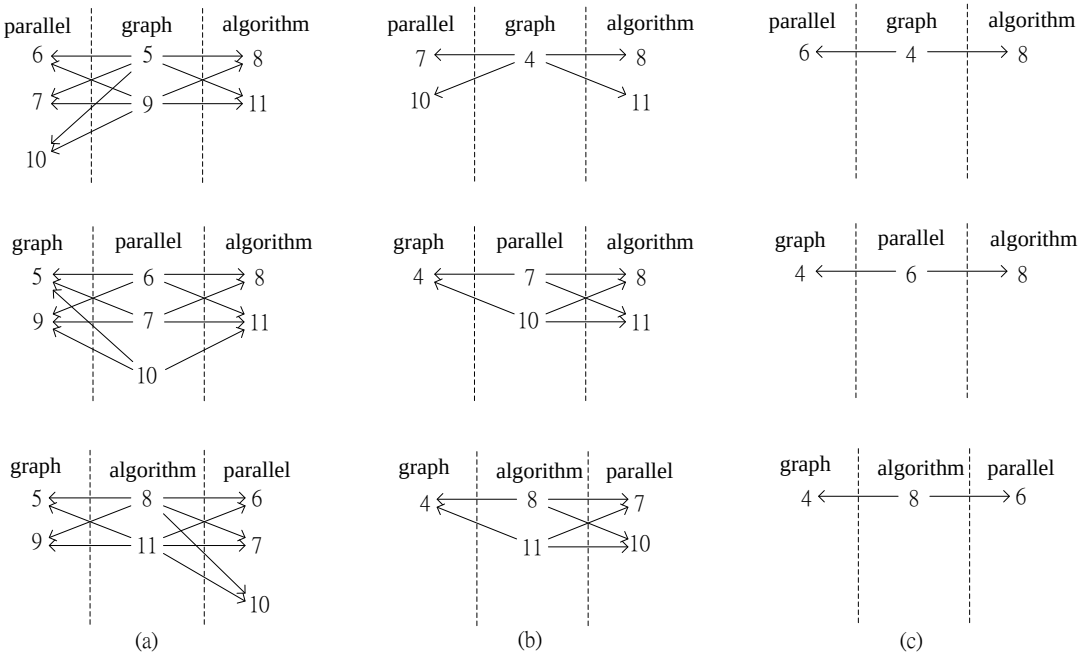


圖 2.15 搜尋子空間各分群至另外兩分群之所有節點組合

如圖 2.14，圖 2.14(a)為第一個搜尋子空間，關鍵字“graph”分群中扣除最佳解中的節點 4，另外兩關鍵字分群節點均保留。圖 2.14(b)為第二個搜尋子空間，關鍵字“graph”分群只留下最佳解的節點 4，關鍵字“parallel”分群中扣除最佳解中的節點 6，關鍵字“algorithm”分群節點均保留。圖 2.14(c)為第三個搜尋子空間，關鍵字“graph”與“parallel”分群只留下最佳解的節點 4 與節點 6，關鍵字“algorithm”分群中扣除最佳解中的節點 11。並在以上三個子空間搜尋並回傳第二名的 *r*-clique 組合，對應如圖 2.15。

接下來搜尋第三、四、五、六、…之 *r*-clique 也依此類推，直到各分群節點無法再被分割為子空間。

top-*k* List
r-clique
4,6,11
5,7,8
7,8,9
9,10,11
6,9,11

圖 2.16Polynomial-Delay 之 top-*k* List

我們在如圖 2.16，Polynomial-Delay 演算法輸出之前 5 名 *r*-clique 為{4, 6, 11}、{5, 7, 8}、{7, 8, 9}、{9, 10, 11}、{6, 9, 11}。參考圖 2.7，與 Branch&Bound 輸出結果(圖 2.4)恰好相同。研究[KA11]中有證明，未搜尋的節點間距離，其最短距離≦2R，顯示其方法的可行性，同時此搜尋法將時間複雜度，改善成為 $O(m^2|S_{MAX}|^2)$(m 為關鍵字個數，$|S_{MAX}|$為搜尋關鍵字分群空間的最大值)。

我們在最後實驗中也應證了此演算法的有效性。但是我們發現，在龐大的圖形資料中使用 Polynomial-Delay 演算法，伴隨 *k* 值得變大，分割的搜尋空間因 *k* 值越變越巨大，需要的計算量也隨之大量增加，因此相當耗時。

於是我們提出將圖形資料轉換成樹狀結構的形式[張 13]，再從樹狀結構中搜尋，來改善效率，並透過實驗證明我們提出的方法的可行性。

第 3 章以樹為基礎的查詢處理做法

由於對資料圖的搜尋計算複雜，加上針對資料樹的搜尋研究已經相當成熟，所以我們採用將本來的資料圖建出樹後，再利用資料樹的搜尋方法快速找出搜尋結果。在資料樹的研究上許多論文是以 SLCA(Smallest Lowest Common Ancestor)搜尋方式作為搜尋結果，但是 SLCA 搜尋方式會過濾掉許多其它有可能也需要被回傳的資訊，並不是相當適合我們的查詢方式。比起 SLCA 搜尋方式 ELCA 搜尋方式能夠保留部分被 SLCA 搜尋方式過濾掉的需要資訊，故我們選擇 ELCA 搜尋方式作為我們將資料圖轉換成資料樹之後的搜尋方式，並且選擇目前所知是最快的方法[BCGL+12]作為本論文使用的搜尋方法。

接下來在本章第一節，介紹與資料樹搜尋相關的基本定義如 LCA、SLCA、ELCA，第二節我們將介紹如何將資料圖轉換成三種資料樹，分別為 HD tree、MB tree 以及 BFS+ tree，第三節介紹我們提出的 SRE 演算法，如何在樹狀結構上以搜尋 ELCA 的方式回傳 *r*-clique 並輸出 top-*k* 組結果。

3.1 基本定義

對於資料樹(如 XML tree)進行關鍵字查詢，已經被廣泛地研究，以下我們介紹幾個常見的定義。

[定義 3.1]：LCA 節點：對樹 T 進行關鍵字查詢 $Q = \{k_1, ..., k_m\}$，子孫中包含一組 k_1 到 k_m 對應節點的層數最低的節點，稱為 Q 對 T 的 LCA 節點。

[定義 3.2]：SLCA 節點：對樹 T 進行關鍵字查詢 $Q = \{k_1, ..., k_m\}$，在 Q 對 T 的 LCA 節點中若其子孫不包含其他 LCA 節點，則稱為 Q 對 T 的 SLCA 節點。

[定義 3.3]：ELCA 節點：對樹 T 進行關鍵字查詢 $Q = \{k_1, ..., k_m\}$，假設某個 LCA

節點 e，在以 e 為 root 的子樹中排除其他 SLCA 的子樹後，對$\{k_1, ..., k_m\}$仍各自包含對應節點，則稱此 LCA 節點 e 為 Q 對 T 的 ELCA 節點。

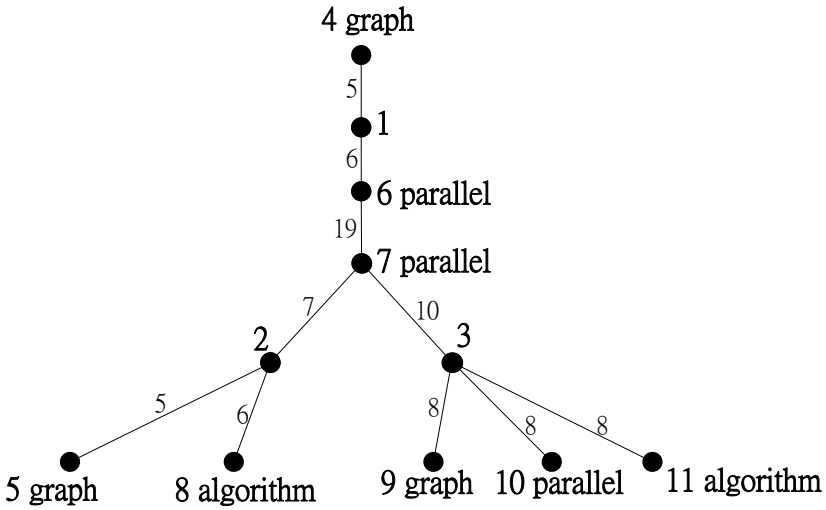


圖 3.1 範例生成樹

圖 3.1 為對應圖 2.1 的一生成樹，以下我們用圖 3.1 來舉例說明定義。

[範例 3.1]：以圖 3.1 的樹作查詢，查詢關鍵字為"parallel", "graph", "algorithm"，則可以得到 LCA 節點為 6, 7, 3，SLCA 節點為 3，進一步依照 ELCA 節點定義可以得到節點 3, 7 二個節點為 ELCA 節點。

[定義 3.4]：ELCA 樹: 對一個樹進行關鍵字查詢 Q，取得 ELCA 節點後將該 ELCA 節點所含有所有的對應節點到該 ELCA 節點中間的路徑取出合成一個新的樹，我們稱此樹為一個 ELCA 樹。

[範例 3.2] ：以圖 3.1 的樹作查詢，查詢關鍵字為"parallel", "graph", "algorithm"，依照範例 3.1 我們可得知 ELCA 樹為{7, 2, 5, 8}和{3, 9, 10, 11}，如圖 3.2 所示。

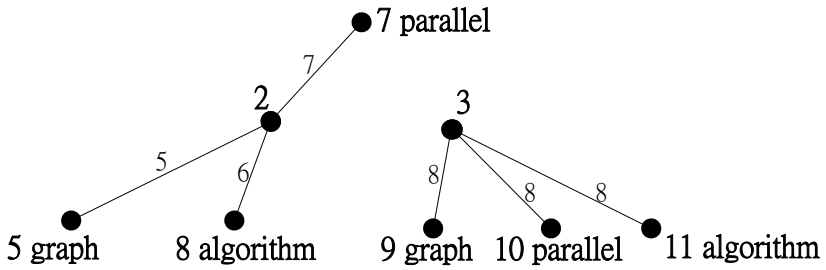


圖 3.2 範例 ELCA tree

由圖範例 3.1 與範例 3.2 為例，也顯示出節點 7 不是 SLCA 節點，假如採取 SLCA 搜尋法，則會遺漏一組我們需要的節點，這也說明我們選擇 ELCA 搜尋法做為我們在樹狀結構中的主要搜尋方法。

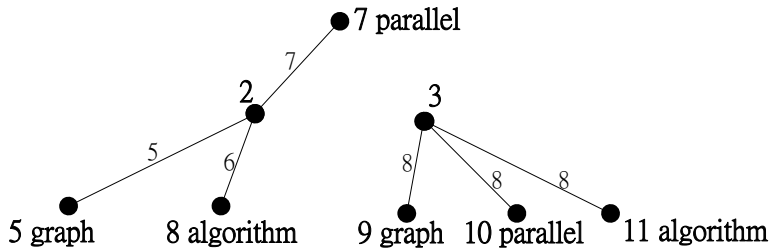


圖 3.3 範例 ER tree

[定義 3.5]：ER tree 和 top-*k* 輸出: 對一個樹進行關鍵字查詢 *Q*，取得 ELCA 樹後，若該樹中一個以 ELCA 節點為根節點的 subtree，包含每個查詢關鍵字正好一次，且節點間的最短距離皆小於 *r*，也就是該組對應節點符合 *r*-clique 定義，形成一個 *r*-clique(參照圖 2.2)，則我們稱此樹為一個 ER tree。而我們將 ER tree 的邊加總為 *weight* 做為我們 top-*k* 排序依據。

[範例 3.3]：參考圖 3.3，查詢關鍵字為“parallel”, “graph”, “algorithm”，且 *r* 值為 20，由節點 7 為根節點的 ER tree 為{7, 2, 5,8}，由節點 3 為根節點的 ER tree 為{3, 9, 10, 11}，最後將其中搜尋到的 *r*-clique，依其在樹上所有邊的 *weight* 總和排序 top-*k*，如圖 3.4。

top-*k* List	
r-clique	*weight*
5,7,8	18
9,10,11	24

圖 3.4 top-*k* List

將圖 3.4 與圖 2.4(Branch&Bound 之 top-*k* 結果)比較，會發現雖然需要計算的 candidute 變少，但輸出的結果也變少，以下探討影響搜尋結果的因素：

節點 1、4、6 一開始就被排出在搜尋 *r*-clique 範圍之外，也因為如此，原本可能成為 *r*-clique 的候選節點組合{4, 6, 11}、{6, 8, 9}、{6, 9, 11}不會被搜尋輸出。此為影響搜尋 *r*-clique 準確率的其中一個因素—即因為節點無法成為 ELCA 樹中的節點，使得與該節點相連的葉節點被排除在搜尋範圍之外，導致可能成為 *r*-clique 的節點組合無法輸出。

另外一個個狀況是，節點分散在兩個不同的 ELCA tree 中，故使原本可能成為 *r*-clique 的候選節點組合{7, 8, 9}、{7, 9, 11}、{8, 9, 10}無法被搜尋輸出。此為另一個影響搜尋 *r*-clique 準確率的個因素—即因為節點分散到不同的 ELCA tree 下，導致可能成為 *r*-clique 的節點組合無法輸出。

因此我們接著提出了不同的轉換樹策略來改善上述範例中提到影響準確率的因素。

3.2 轉換樹策略

以下分節討論我們所使用的轉換策略。這些策略的完整演算法請參見[張 13]，在此我們僅介紹大概的步驟，並給予範例說明。

3.2.1 Highest Degree Tree (HD tree)

這個策略優先選擇 *degree* 較高的點來建出較平坦的樹，希望能以較平坦的樹產生層數較低且總數量較少的 ELCA 樹，盡量減少原本屬於同一個 *r*-clique 的對應節點因分散到不同 ELCA 樹下而無法被回傳。由於在建樹過程中需要知道每個節點的 *degree*，故在建樹之前我們會先對資料圖中所有節點先計算 *degree* 的資料，並用一個陣列記錄這些節點資料，同時在陣列中記錄各個節點是否已經被加入到樹中。建樹的步驟基本為：以最高 *degree* 的節點為根節點，記錄根節點所連接到的所有節點作為根節點的小孩節點，再以剛剛所記錄的節點中擁有最高 *degree* 節點的點繼續搜尋可連結到的節點，直到圖中所有節點皆被加入轉換樹中。

ID	1	2	3	4	5	6	7	8	9	10	11
degree	3	3	6	1	2	4	4	3	2	2	3

表 3.1 各節點的 *degree*

ID	3	6	7	1	2	8	10	11	5	9	4
degree	6	4	4	3	3	3	3	3	2	2	1

表 3.2 各節點的 *degree* (依 *degree* 高到低排序)

接著我們用圖 2.1 作為範例來解釋建構 HD tree，表 3.1 及表 3.2 列出了各個節點的 *degree*。首先會從圖中選出 *degree* 最高的點作為 HD tree 的根節點，在圖中節點 3 為最高的點，將節點 3 作為 HD tree 的根節點並加入和節點 3 有連結的點{6, 7, 8, 9, 10, 11}作為子節點，並將這些節點依照 *degree* 的高低放入一個佇列中排序，接著從佇列中取出 *degree* 最高的點作為下一個

要加入轉換樹的點，在此處取出的點為節點 6，並將和節點 6 有連接且尚未被加入到轉換樹的點加到轉換樹中並放入佇列排序，之後重複此動作直到圖 2.1 的所有節點皆被加入到轉換樹中，最後會得到圖 3.5 的轉換樹。

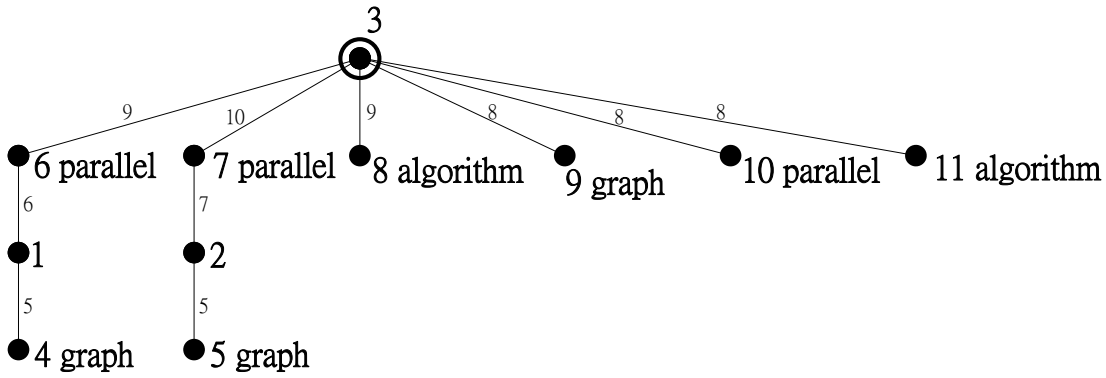


圖 3.5 範例查詢圖所生成的 HD tree

如圖 3.5，若現在下達查詢關鍵字“graph”, “parallel”, “algorithm”，我們可以得到 ELCA 節點為節點 3，在考慮 *r* 值為 20 的情況下，我們可以列出能夠成為 *r*-clique 的節點組合有{6, 8, 9}、{6, 9, 11}、{7, 8, 9}、{7, 9, 11}、{8, 9, 10}、{9, 10, 11}六組。

3.2.2 Most-branch Tree (MB tree) ：

此策略的目的是希望提高樹的分支個數，希望藉以提高 ELCA 節點的個數，也就是 ELCA 樹的個數，進而回傳更多的解。以下為大略的建樹步驟：

先對欲搜尋圖作 Breadth first search 建出一轉換樹，再將圖中剩下未使用的邊試著替換轉換樹的邊，如果能產生較多分支節點的樹則以新邊取代舊邊，同時以新轉換樹為基準繼續檢查是否有可替換邊能產生更多分支節點。直到產生該圖的最多分支轉換樹。

在此我們一樣以圖 2.1 作為範例說明 MB tree 的建立，首先我們必須先對原

始圖進行 Breadth-first search 建構出 BFS tree，如圖 3.6 所示，此時具有分支的節點數為 3，接著我們一一檢查沒有被加入到轉換樹內的邊，先選一個邊加入到目前的轉換樹中產生有一個環(cycle)的圖，接著再將環上的每個邊檢查，看拿掉一個邊之後的轉換樹是否比本來的轉換樹擁有更多分支點(Branch node)，若有則將有更多分支點的生成術取代本來的轉換樹，之後再繼續檢查其他的邊，直到可以檢查完所有邊為止，比較圖 3.6 及圖 3.7 我們可以發現邊{8, 11}被加入到轉換樹中而邊{1, 11}被剔除，導致節點 8 也變成分支節點。最後即可得到如圖 3.7(a) 的 MB tree，其分支數是 4，大於原本圖 3.6 的分支數 3。

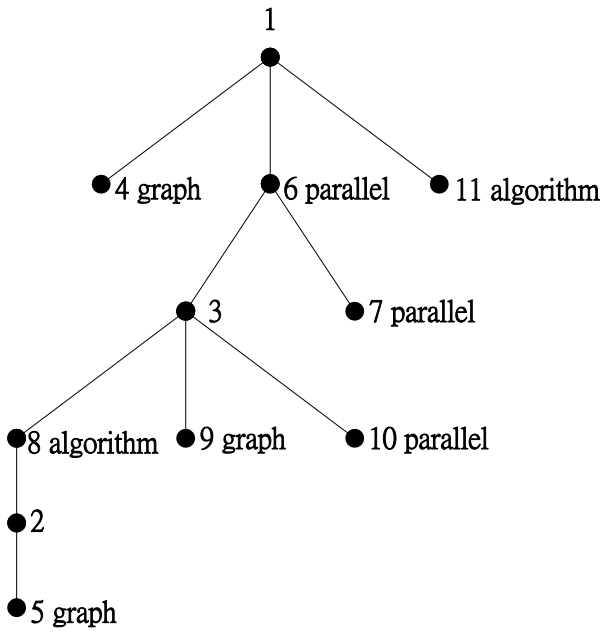
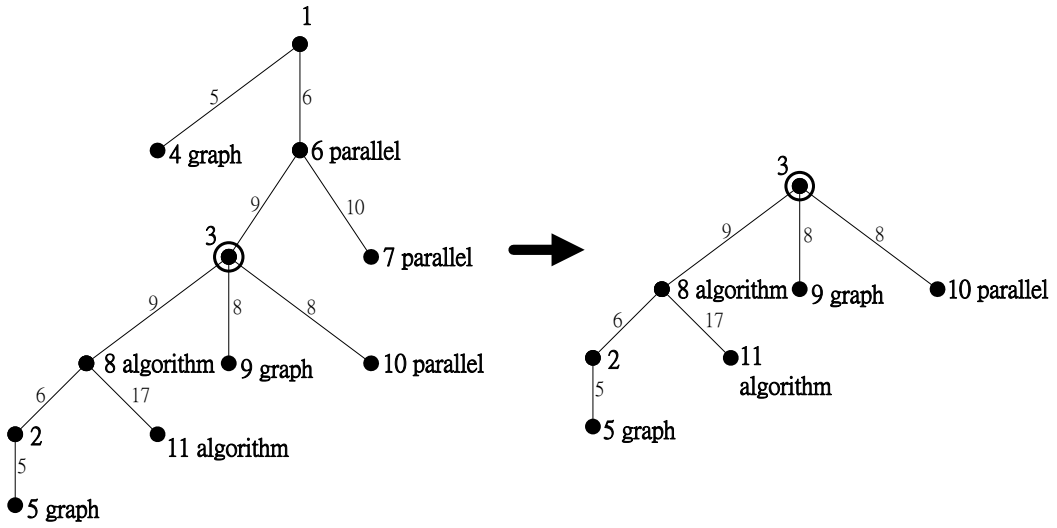


圖 3.6 範例查詢圖所生成的 BFS tree



(a) (b)

圖 3.7 (a)範例查詢圖所生成的 MB tree (b)MB tree 對應之 ELCA tree

若針對圖 3.7，下達查詢關鍵字“graph”, “parallel”, “algorithm”，我們可以得到 ELCA 節點為節點 3，其 ELCA tree 如圖 3.7(b)，在考慮 *r* 值為 20 的情況下，我們可以列出能夠成為 *r*-clique 的節點組合為{8, 9, 10}。

3.2.3 Breath First Search+ tree (BFS+ tree) ：

為了避免在搜尋時無法考慮到沒有被畫入轉換樹的邊而找不到 *r*-clique，我們希望加入沒有被畫入轉換樹的邊並產生新的節點以解決這個問題。我們參考[CCW13]的做法，以下則為建樹的大略步驟：

先對欲搜尋圖作 Breath first search 建出一轉換樹，再將圖中剩下未使用的邊依照節點編號的大小，找到該條邊在轉換樹中編號較小的節點，在該點加上編號較大的點作為新的子節點。

我們再以圖 2.1 作為範例，一開始一樣必須先作 Breadth-first search 建構出圖 3.6 的 BFS tree，之後將沒有被加入到 BFS tree 中的邊以加入新子節點的方式一一加入到 BFS tree 中，最後可以得到圖 3.8 的 BFS+ tree，在圖 3.8 中，標示虛線的邊即為一開始沒有被加入到 BFS tree 中的邊。此種建立方式讓 BFS+ tree 能

夠補強另外兩種轉換樹無法考慮到未加入到轉換樹中的邊資訊的狀況。由於將圖 2.1 中所有的邊都加入到樹中，圖 3.8 的 BFS+ tree 的節點數量比起圖 2.1 的資料圖多出了 8 個。

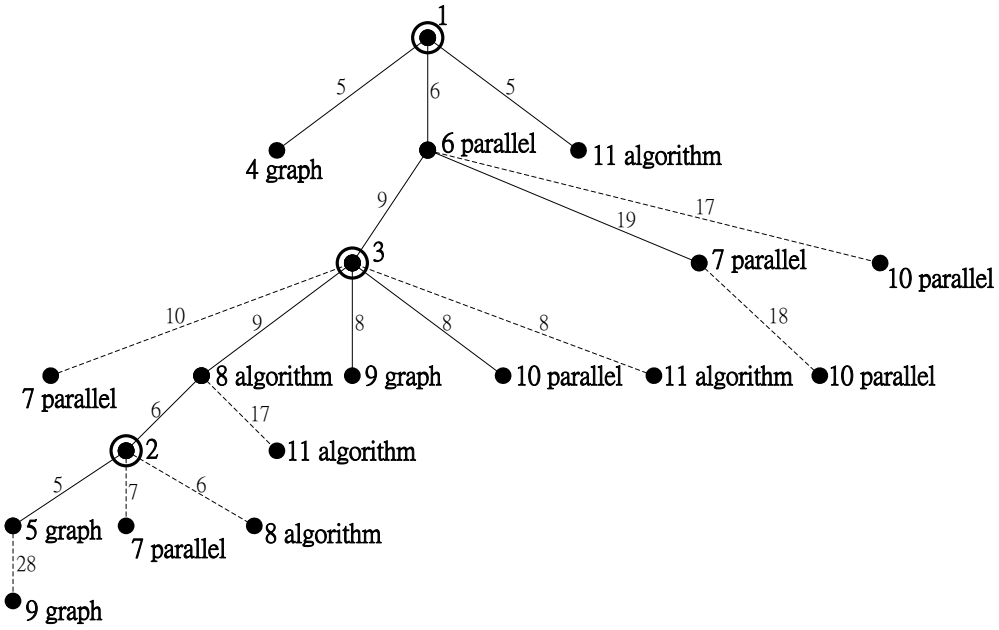


圖 3.8 BFS+ tree

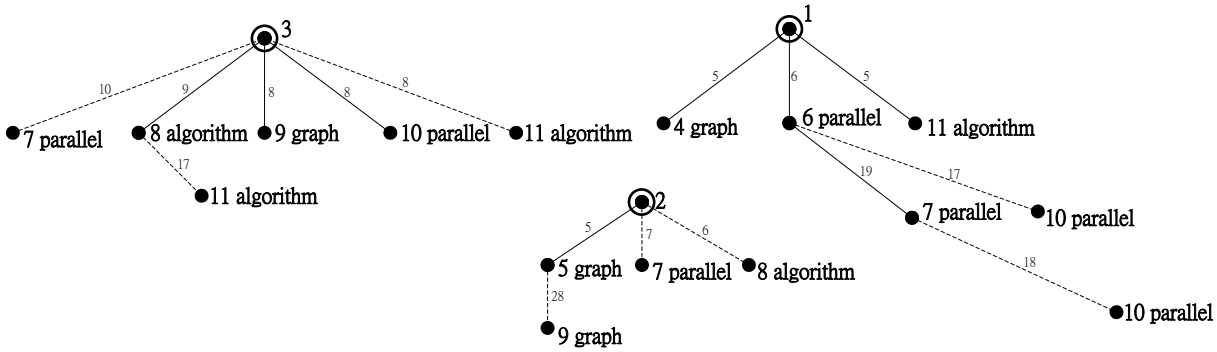


圖 3.9 BFS+ Tree 分割為三個 ELCA tree

若現在下達查詢關鍵字”graph”, ”parallel”, ”algorithm”，在考慮 *r* 值為 20 情況下，BFS+ tree 會被分割成三個 ELCA tree（如圖 3.9）：以節點 1 為根節點的 ELCA tree，我們列出能夠成為 *r*-clique 的節點組合為{4, 6, 11}；以節點 2 為根節點的 ELCA tree，我們列出能夠成為 *r*-clique 的節點組合為{5, 7, 8}；以節點 3 為根節點的 ELCA tree，我們列出能夠成為 *r*-clique 的節點組合有{7, 8, 9}、{7, 9, 11}、

{8, 9, 10}、{9, 10, 11}。

總合而言，根據上述策略，我們可以產生不同特性的轉換樹：(1) HD tree 以高 *degree* 為優先建出較平坦的樹，(2) MB tree 建出分支最多的樹以期待能找出最多組 ELCA 結果，(3) BFS+ tree 將前述四種策略未考慮到的邊再加到轉換樹中，避免轉換樹無法考慮到未加入邊的資訊。在此例中，雖然 HD tree 產生的 *r*-clique 最多，MB tree 產生的 *r*-clique 最少，但在第 4 章中，我們將以更多的實驗來比較此三種方法的優劣。

總結以上提出的搜尋方法，在最後一節中，說明我們如何在樹狀結構中搜尋輸出前 *k* 名 *r*-clique。

3.3 查詢資料樹演算法

以下列出我們的 SRE(Search R-clique based on ELCA)演算法：

演算法名稱：SRE 輸入：資料樹 T, 關鍵字$\{k_1, k_2, ..., k_m\}$共 m 個關鍵字, 距離限制 r, top-k 的 k 值 輸出：所有符合搜尋的前 k 名 r-clique
L01: $E_S \leftarrow$ FindELCAset(T, k_1, …, k_m) L02: for each ELCA tree E_A in E_S L03: $\{M_1, M_2, \ldots, M_m\} \leftarrow$ all match keyword nodes in E_A L04: RCset.add (BABR(M_1, M_2,…,M_m ,r)) L05: for each r-clique RC in RCset L06: ERtreeSet.add (GetERtree(RC)) L07: Top-kList ←rank(ERtreeSet, k) L08: return Top-kList

圖 3.10 SRE (Search R-clique based on ELCA)演算法

SRE (Search R-clique based on ELCA)演算法，運用 Branch&Bound 演算法的概念，逐一檢查樹中兩兩對應節的的距離是否小於 *r*。SRE 演算法大略可分為下

列兩個階段：

1. 對轉換樹 T 找出所有的 ELCA 樹

2. 對每個 ELCA 樹，套用修正的 Branch&Bound 演算法(BABR)來尋找所有的 R-clique，並將 BABR 演算法找到的所有 *r*-clique 集中到一個集合中，之後呼叫 GetERtree 得到每個 *r*-clique 的 ERtree 作為搜尋 top-*k* 組結果回傳。

如圖 3.10，開始處理查詢之後我們在 L01 對轉換樹 T 進行 ELCA 搜尋得到多個 ELCA 樹，並將所有 ELCA 樹存入集合 E_S 中準備作後續的判斷。從 L02 開始我們對 ELCA 樹集合 ES 開始作 *r*-clique 的判斷， L04 的 BABR 即是修改過後的 Branch&Bound 演算法，將所有對應節點集合和限制 *r* 套用 Branch&Bound 演算法搜尋出存在 E_A 裡的 *r*-clique，再將找到的 *r*-clique 存入 RCset 中，在 L05 及 L06 我們將找到的 *r*-clique 呼叫 GetERtree 函式得到該 *r*-clique 的 ER tree 並存入 ERtreeSet 中，確定已經檢查完所有 ELCA 樹並得到所有 ER tree 之後，將 ER tree 集合 ERtreeSet 取得各個 RE tree 的 *weight*(即 tree 的 edge 總和)，在 L07 套入如圖 2.5 L04 行的 rank 演算法排序完畢後，依照輸入的 *k* 值輸出 top-*k*List。

我們以前一節提出的三種轉換樹為例，說明如何透過 SRE 演算法在樹狀結構中搜尋 *r*-clique，並回傳 top-*k* 組結果。

延續之前的範例首先以 HD tree 圖 3.5 為例，執行 SRE 演算法，會得到 6 個 *r*-clique，而其對應的 ER tree，如圖 3.11：

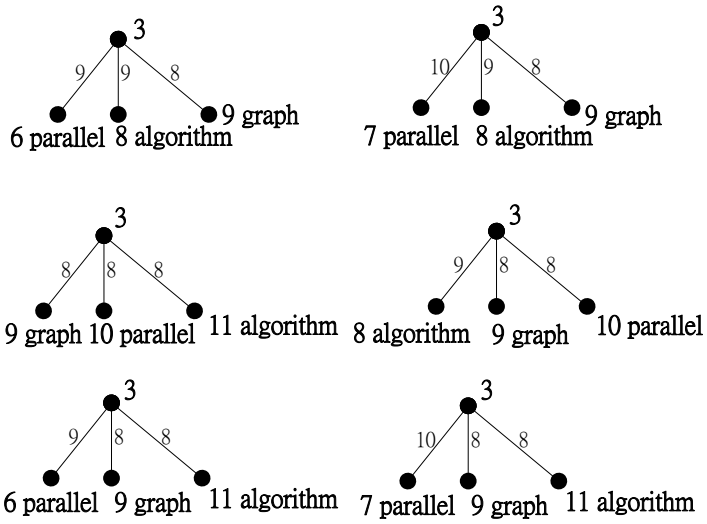


圖 3.11 HD tree 之 ER tree

top-*k* List	
r-clique	*weight*
9,10,11	24
6,9,11	25
8,9,10	25
6,8,9	26
7,9,11	26
7,8,9	27

圖 3.12 以 SRE 搜尋 HD tree 之 top-*k* List

top-*k* List	
r-clique	*weight*
4,6,11	16
5,7,8	18
7,8,9	21
9,10,11	24
6,9,11	25
8,9,10	25
6,8,9	26
7,9,11	26

B&B top-*k*

top-*k* List	
r-clique	*weight*
9,10,11	24
6,9,11	25
8,9,10	25
6,8,9	26
7,9,11	26
7,8,9	27

HD tree top-*k*

圖 3.13 HD tree 與 Branch&Bound 搜尋結果對照

注意到 ER tree 兩節點的邊大小，為兩節點實際在 ELCA 樹中的距離。接著我們將 ER tree 的 *weight*(所有 edge 的總和)做 top-*k* 排序，如圖 3.12。*k* 值設定為 5，所以只輸出前 5 組 *r*-clique，分別為{9,10,11}、{6,9,11}、{8,9,10}、{6, 8, 9}、{7, 9, 11}。圖 3.13 為 HD tree 搜尋結果(圖 3.12)與 Branch&Bound 演算法搜尋結果(圖 2.4)的比較，可以看出 HD tree 輸出的前 5 個 *r*-clique，其 *weight* 總和略高於 Branch&bound 的輸出。

接著以 MB tree，圖 3.7(a)為例，延續之前的範例執行 SRE 演算法，可得到一組 r-clique 如圖 3.7(b)，而其對應的 ER tree 如圖 3.14：

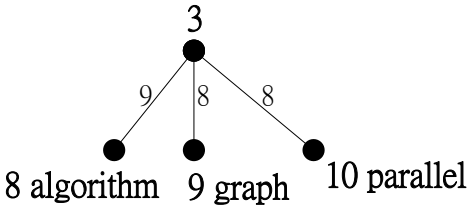


圖 3.14 MB tree 之 ER tree

top-*k* List	
r-clique	*weight*
8,9,10	25
Ø	Ø
Ø	Ø
Ø	Ø
Ø	Ø

圖 3.15 以 SRE 搜尋 MB tree 之 top-*k* List

top-*k* List	
r-clique	*weight*
4,6,11	16
5,7,8	18
7,8,9	21
9,10,11	24
6,9,11	25
8,9,10	25
6,8,9	26
7,9,11	26

B&B top-*k*

top-*k* List	
r-clique	*weight*
8,9,10	25
Ø	Ø
Ø	Ø
Ø	Ø
Ø	Ø

MB tree top-*k*

圖 3.16 MB tree 與 Branch&Bound 搜尋結果對照

將此組 *r*-clique 輸出對應的 ER tree，如圖 3.14，ER tree 兩節點的邊大小，為兩節點實際在 ELCA 樹中的距離。*k* 值設定為 5，但只有一個搜尋結果，所以只輸出 *r*-clique{8, 9, 10}，如圖 3.15。圖 3.16 為 MD tree 搜尋結果(圖 3.15)與 Branch&Bound 演算法搜尋結果(圖 2.4)。

最後以 BFS+ tree(圖 3.8)為例，執行 SRE 演算法，延續之前的範例，可以看到 BFS+ tree 被分割為三個 ELCA Tree 如圖 3.9。

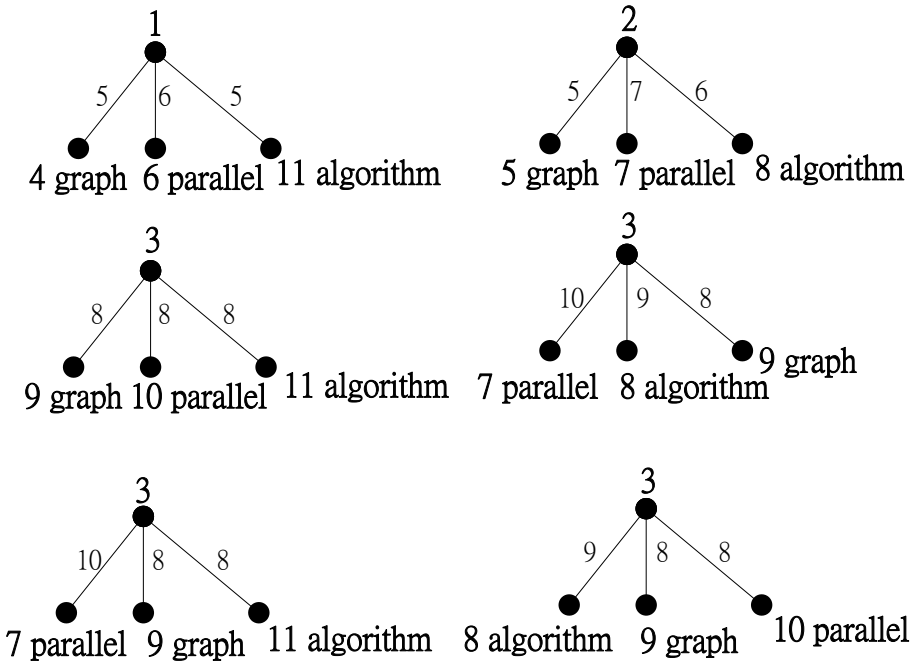


圖 3.17 以節點 1、2、3 為根節點的 ER tree

top-*k* List	
r-clique	*weight*
4,6,11	16
5,7,8	18
9,10,11	24
8,9,10	25
7,9,11	26
7,8,9	27

圖 3.18 BFS+ tree 之 top-*k* List

top-*k* List	
r-clique	*weight*
4,6,11	16
5,7,8	18
7,8,9	21
9,10,11	24
6,9,11	25
8,9,10	25
6,8,9	26
7,9,11	26

B&B top-*k*

top-*k* List	
r-clique	*weight*
4,6,11	16
5,7,8	18
9,10,11	24
8,9,10	25
7,9,11	26
7,8,9	27

BFS+ tree top-*k*

圖 3.19 BFS+ tree 與 Branch&Bound 搜尋結果對照

將所得之 *r*-clique 輸出對應的 ER tree，如圖 3.17，ER tree 兩節點的邊大小，為兩節點實際在圖 3.9 之 ELCA 樹中的距離。接著我們將 ER tree 的 *weight*(所有 edge 的總和)做 top-*k* 排序，如圖 3.18。*k* 值設定為 5，故輸出前 5 組 *r*-clique，分別為{4, 6, 11}、{5, 7, 8}、{9, 10, 11}、{8, 9 , 10}、{7, 9, 11}。圖 3.19 為 BFS+ tree 搜尋結果(圖 3.18)與 Branch&Bound 演算法搜尋結果(圖 2.4)，對照發現在前 5 名之結果中命中 3 組。

注意到雖然此三種方法不能完全輸出 top-*k* 的結果，但我們提出的方法將複雜度為 $O(m^2 |C_{MAX}|^{m+1})$的 Branch&Bound 演算法中變數$|C_{MAX}|$，改善為 ELCA tree 的範圍，因此降低了搜尋與計算的時間。我們將透過實驗，與原本 Branch&Bound 演算法以及研究[KA11]提出的 polynomial-delay 演算法針對有效性和效率做比較。

第 4 章實驗

在本章節中，以我們提出的三個樹狀結構透過 SRE(Search R-clique based on ELCA)演算法搜尋的方法與 Branch&Bound、Polynomial-Delay 演算法來比較效率和查詢有效性。首先說明我們進行實驗的環境，我們以個人電腦作為實驗的環境，其 CPU 為四核心的 Intel i7 2600，其中每個核心的時脈為 3.4GHz，記憶體為 16GB，所採用的作業系統為 Windows 7 企業版 SP1。

找出 ELCA 樹是採用[ZBWL+12]的做法，而實驗的資料集為 2008-2013 年 DBLP 網站蒐集的文獻資料，其中的作者、論文和引用資訊，圖中節點總數為 678620，邊為 3415726 個。而邊距離的計算方式我們參照[KA11]的定義，假設有兩相鄰節點 u、v，則 u 到 v 的距離為$(\log_2(1 + u_{deg}) + \log_2(1 + v_{deg}))/2$，其中 u_{deg} 為 u 的 degree，v_{deg} 為 v 的 degree。

如同 Branch&Bound 演算法與 Polynomial-Delay 演算法，我們會根據原始資料圖(如圖 2.1)建立所有節點的鄰居索引—以記錄兩兩節點間的最短距離，與建立反轉索引(inverted list)—以紀錄一個關鍵字和含有該關鍵字的節點編號。針對前者，目前我們是利用 Dijkstra 演算法求出指定節點到圖中其他節點的最短距離與最短路徑，而且距離太遠的點不予記錄以節省空間。

以下我們列出實驗使用的查詢句以及關鍵字的出現頻率：

出現頻率	關鍵字
0.0001~0.0003	genes,trees,paths
0.0003~0.0006	protocol,fast,retrieval
0.0006~0.0009	programming , scheme,scheduling,environments,parallel
0.0009~0.0012	framework , modeling,problem, methods,selection,sensor
0.0012~0.015	optimization, dynamic, modeling, problem
0.015~	cloud, network, control, detection, performance

表 4.1 關鍵字出現頻率

編號	資料集	關鍵字
TQ1	DBLP	gene, trees
TQ2	DBLP	protocol,fast,retrieval
TQ3	DBLP	programming,parallel,environments
TQ4	DBLP	scheduling,hybrid,communication
TQ5	DBLP	methods,selection,sensor,imperfect
TQ6	DBLP	optimization,dynamic,framework
TQ7	DBLP	network,control,modeling
TQ8	DBLP	network,detection,performance
TQ9	DBLP	cloud,modeling,problem

表 4.2 查詢句 TQ1 ~ TQ9

我們在表 4.1 列出查詢句會用到的關鍵字在 DBLP 當中出現的頻率，大致分為低中高，0.0001~0.0006 為低，0.0006~0.0012 為中，大於 0.0012 則為高。接著表 4.2 是我們所使用的查詢句，查詢句 TQ1~TQ9 主要是依照關鍵字的出現頻率來分群，查詢句 TQ1~TQ2 為頻率低的關鍵字組合，查詢句 TQ3~TQ5 為頻率中的關鍵字組合，查詢句 TQ6~TQ9 為頻率高的關鍵字組合。

4.1 有效性的評估

以下我們列出以前述的查詢句所作出的實驗結果比較有效性，也就是判斷我們的做法是否能有效地找出前 *k* 個 *weight* 總和最小的 *r*-clique，更明確地說，我們以 Branch&Bound 演算法所找出的第 *k* 個 *r*-clique 在 Steiner tree 中對應的 *weight* 總和為分子，我們提出的 SRE 演算法在三種樹狀結構搜尋的方法的 *weight* 總和為分母。以及以 Branch&Bound 演算法所找出的第 *k* 個 *r*-clique 在 Steiner tree 中對應的 *weight* 總和為分子，以 Polynomial-Delay 演算法輸出的第 *k* 個 *r*-clique 在 Steiner tree 中對應的 *weight* 總和為分母。而我們將 Branch&Bound 的有效性設定為 100%。以下為算式：

$$\text{我們方法的有效性(\%)} = \frac{\text{第}k\text{個 B\&}B\text{輸出的 }weight\text{總和}}{\text{第}k\text{個 SRE 輸出的 }weight\text{總和}} \quad ,$$

$$\text{Polynomial-Delay 演算法有效性(\%)} = \frac{\text{第}k\text{個 B\&}B\text{輸出的 }weight\text{總和}}{\text{第}k\text{個 Polynomial }-\text{Delay 輸出的 }weight\text{總和}}$$

為了讓查詢結果適合人工判別召回率，我們試著將輸出的答案組數盡量控制在數十組到兩百組之間，以下我們將 *r* 設定大小為 7，且 *k* 值設為 5：

查詢句	BFS+ Tree	MB Tree	HD Tree	POLY-D	B&B
TQ1	60.47%	58.96%	58.96%	72.07%	100%
TQ2	75.00%	75.00%	75.00%	91.31%	100%
TQ3	79.18%	79.18%	79.18%	75.23%	100%
TQ4	72.94%	72.94%	65.34%	71.94%	100%
TQ5	61.79%	61.79%	0.00%	81.61%	100%
TQ6	60.98%	60.98%	50.32%	60.89%	100%
TQ7	70.99%	76.69%	71.72%	67.64%	100%
TQ8	71.66%	70.91%	66.49%	67.76%	100%
TQ9	78.58%	76.77%	75.21%	69.80%	100%

表 4.3 查詢有效性(%)

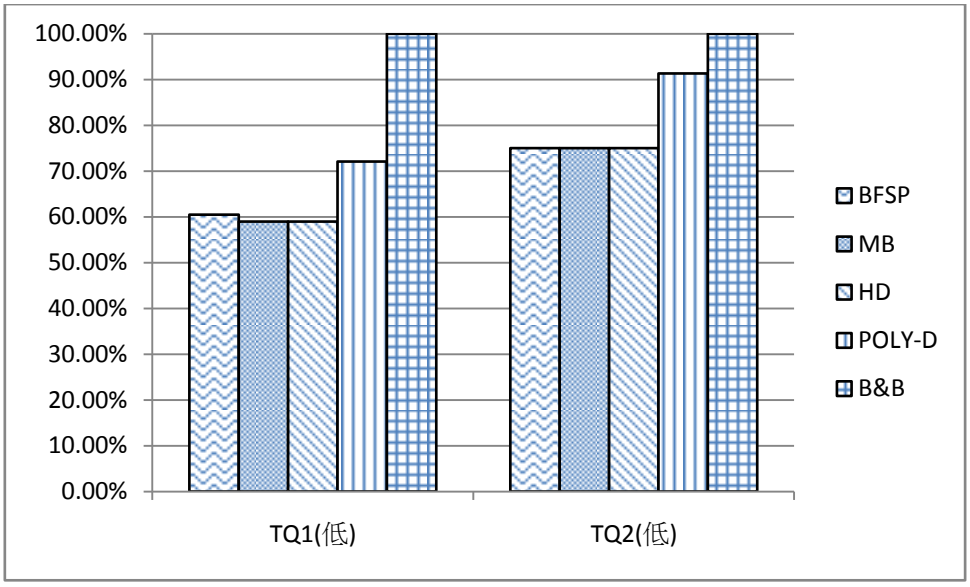


圖 4.1 TQ1~TQ2 查詢有效性(%)比較

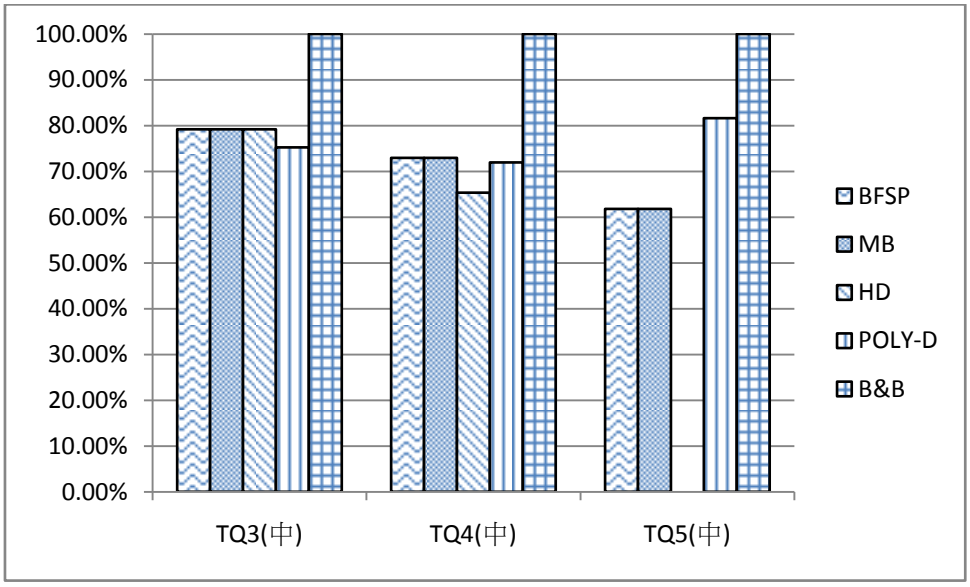


圖 4.2 TQ3~TQ5 查詢有效性(%)比較

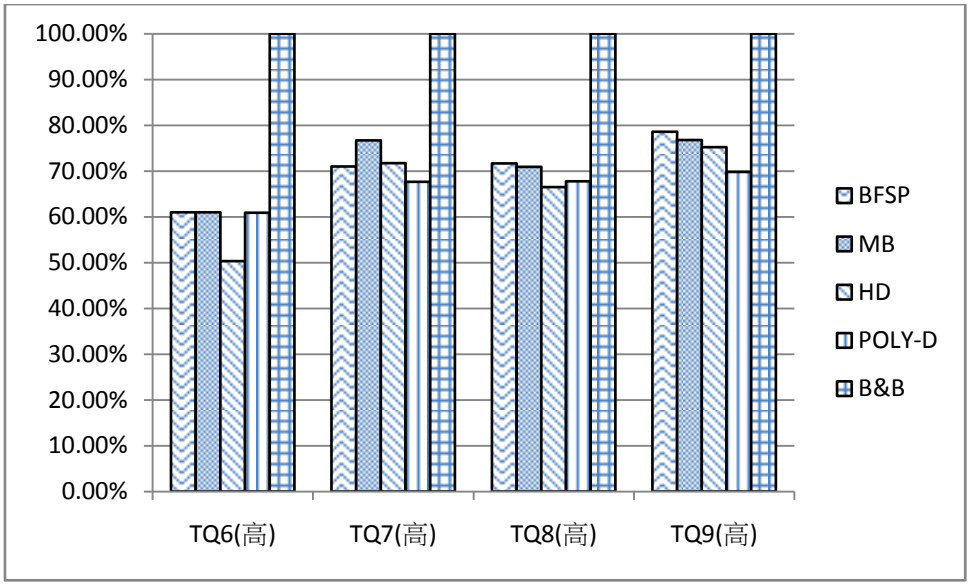


圖 4.3 TQ6~TQ9 查詢有效性(%)比較

圖 4.1~圖 4.3，X 軸為查詢句，Y 軸為根據前式所算出的比率。透過我們提出的三種樹狀結構以及 SRE 搜尋 top-5 *r*-cliques 與研究[MA2011]提出的 Polynomial-Delay 方法比較。在圖 4.1 在關鍵字出現頻率較低的查詢句 TQ1~TQ2 可看出我們的方法表現較差。但是關鍵字出現頻率較高查詢句，因為含有關鍵字節點較多，所轉換而成的 ELCA 樹，節點、邊、分支也較多，因此搜尋範圍變廣，所以在查詢句 TQ6~TQ9 中，我們提出的方法表現上皆優於 Polynomial-Delay。查詢句 TQ5 中，HD Tree 為 0，原因為節點被分割到不同的樹中所以無法找到 *r*-clique 回傳。至於我們的方法中，觀察到以 HD tree 的表現較不好。

4.2 查詢效率評估

接著我們以查詢句 TQ1~TQ9 來比較搜尋效率，在這邊我們將 r 設為 7 且 k 值設定為 5：

查詢句	BFS+ Tree	MB Tree	HD Tree	POLY-D	B&B
TQ1	0.336(s)	0.29(s)	0.131(s)	0.35(s)	0.324(s)
TQ2	3.366(s)	1.162(s)	0.781(s)	4.02(s)	3.228(s)
TQ3	5.482(s)	5.34(s)	0.642(s)	5.737(s)	7.556(s)
TQ4	12.537(s)	1.971(s)	1.716(s)	12.187(s)	11.798(s)
TQ5	15.117(s)	4.122(s)	2.246(s)	13.251(s)	13.294(s)
TQ6	21.389(s)	20(s)	1.855(s)	24.953(s)	28.012(s)
TQ7	38.849(s)	40.54(s)	3.41(s)	63.041(s)	65.556(s)
TQ8	31.402(s)	30.42(s)	2.158(s)	45.874(s)	55.663(s)
TQ9	4.707(s)	0.87(s)	0.802(s)	5.086(s)	7.751(s)

表 4.4 查詢效率(second)

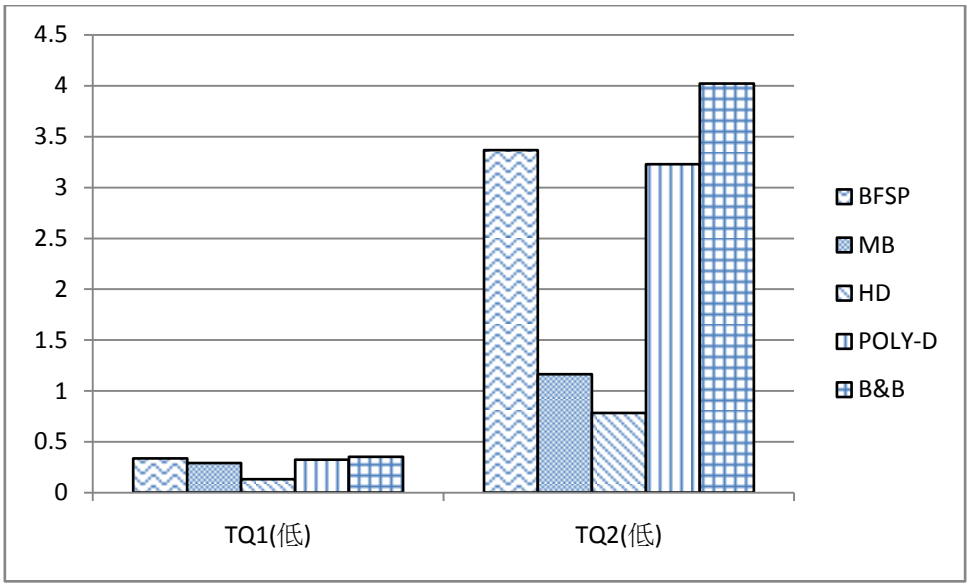


圖 4.4 TQ1~TQ2 查詢效率比較

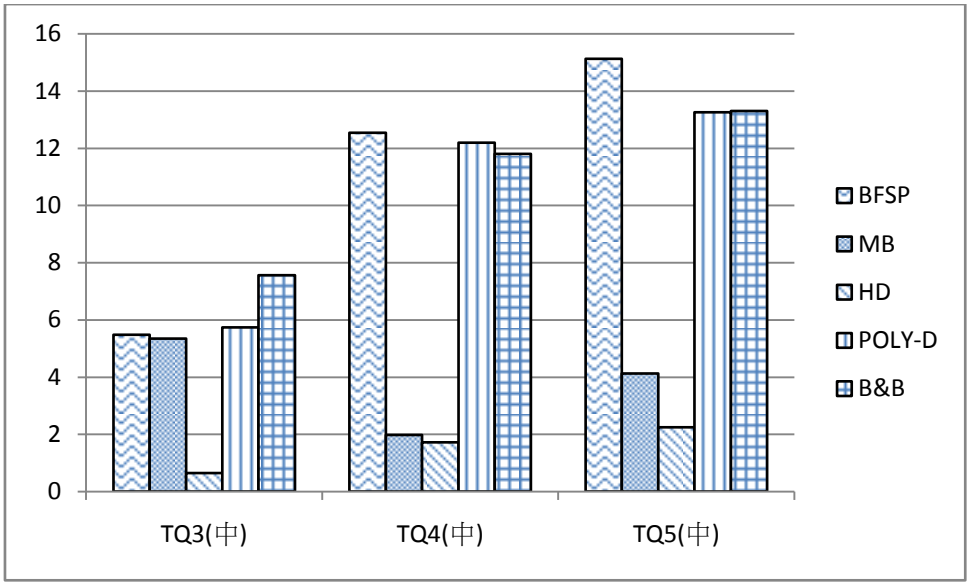


圖 4.5 TQ3~TQ5 查詢效率比較

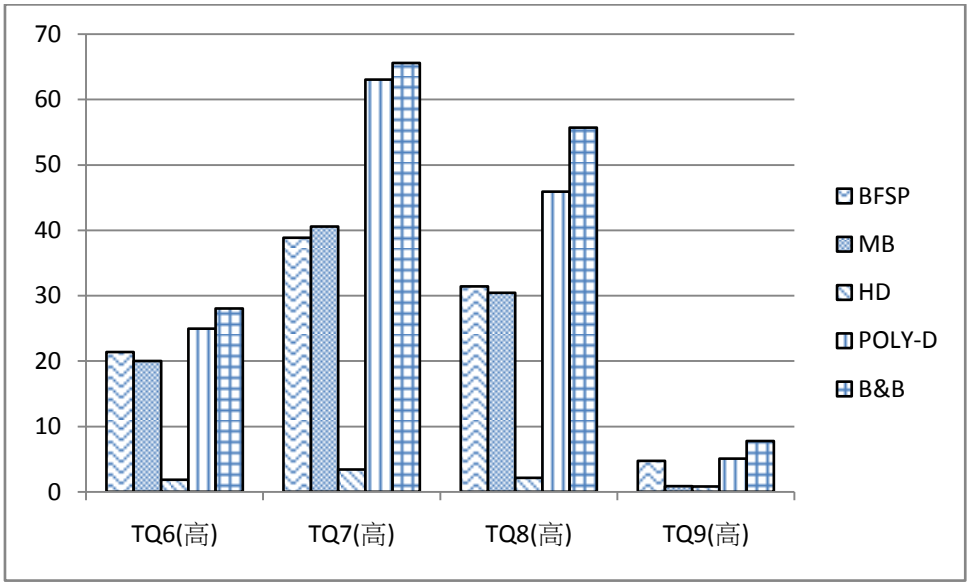


圖 4.6 TQ6~TQ9 查詢效率比較

圖 4.4~圖 4.6，X 軸為查詢句，Y 軸為執行秒數(second)。圖 4.4，關鍵字出現頻率較低的查詢句 TQ1~TQ2 中，BFS+ tree 與 Polynomail-Delay 以及 Branch&Bound 的時間表現相差不多，但 MB tree、HD tree 的時間就明顯低於 Polynomial-Delay 與 Branch&Bound 演算法。圖 4.5 關鍵字出現頻率中等的查詢

句 TQ3~TQ5 中，也是 BFS+ tree 與 Polynomial-Delay 以及 Branch&Bound 的時間表現相差不多，MB tree、HD tree 的時間就明顯低於 Polynomial-Delay 與 Branch&Bound 演算法。圖 4.6 關鍵字出現頻率較高的查詢句 TQ6~TQ9 中，我們的方法在執行時間的表現上明顯低於 Polynomial-Delay 與 Branch&Bound 演算法。其原因為因為關鍵字節點增加，Polynomial-Delay 與 Branch&Bound 演算法，依關鍵字分群的空間也隨之增大，而我們提出的方法只將搜尋空間限制在 ELCA 樹的範圍下，也因此我們提出的方法在時間效率上能優於 Polynomial-Delay 與 Branch&Bound 的方法。在實驗中也能觀察到，BFS+ tree 在時間效率上表現較其他兩種 tree 差，其原因也是因為 tree 的邊與節點都較另外兩種多，所以計算量也較多。

第 5 章結論及未來方向

在本論文中，我們提出了在三種樹狀結構與 SRE 演算法建立搜尋 top-*kr*-cliques 的方法，並且透過實驗應證我們方法的可行性，同時顯示在執行時間的效率表現上，也有明顯提升。尤其是在關鍵字出現頻率較高的查詢句情況下，我們提出的 BFS+ tree 與 MB tree 樹狀結構能維持高的有效性，執行時間的表現更優於 Polynomial-Delay 與 Branch&Bound 演算法。

本論文未來的研究方向，希望能提出更好的轉換樹方法來改善查詢效率及召回率，加強在關鍵字較低頻率的查詢句情況下，搜尋的有效性。最後，本論文提出的 top-*k* 排序方法只是基本的排序法，希望未來能提出更好的 top-*k* 方法來改善查詢效率。

參考文獻

[CCW13] An-Chiang Chu, Kun-Mao Chao, Bang Ye Wu, “A linear-time algorithm for finding an edge-partition with max-minratio at most two.”Discrete Applied Mathematics, 161(7-8): 932-943, 2013.

[DLQW+07] Bolin Ding, Xuemin Lin, Lu Qin, Shan Wang, Jeffrey Xu Yu, Xiao Zhang, “Finding Top-*k* Min-Cost Connected Trees in Databases”, In Proceedings of the ICDE conference, 2007.

[KA11] Mehdi Kargar, Aijun An, “Keyword Search in Graphs: Finding *r*-cliques”, InProceedings of the VLDB Conference, 2011.

[LOF+08] Guoliang Li, Beng Chin Ooi, Jianhua Feng, Jianyong Wang, Lizhu Zhou, "EASE: An Effective 3-in-1 Keyword Search Method for Unstructured, Semi-structured and Structured Data", In Proceedings of the SIGMOD conference, 2008.

[PX07] Yannis Papakonstantinou, Yu Xu, “Efficient LCA based Keyword Search in XML Data”, In Proceedings of the EDBT conference, 2008.

[QYCT09] L. Qin, J. Yu, L. Chang, Y. Tao, “Queryingcommunities in relational databases”, In Proceedings of the ICDE conference, 2009.

[TJM+08] Partha Pratim Talukdar, Marie Jacob, Muhammad Salman Mehmood,Koby Crammer, Zachary G. Ives, Fernando Pereira, Sudipto Guha,"Learning to Create Data Integrating Queries", In Proceeding of theVLDB conference, 2008.

[TPSS11] Yufei Tao, Stavros Papadopoulos, Cheng Sheng, Kostas Stefanidis, “Nearest Keyword Search in XML Documents”, In Proceedings of the SIGMOD Conference, 2011.

[XP05] Yu Xu, Yannis Papakonstantinou, “Efficient Keyword Search for

Smallest LCAs in XML Databases",In Proceedings of the SIGMOD Conference, 2005.

[YCHH12] Junjie Yao, Bin Cui, Liansheng Hua, Yuxin Huang, "Keyword Query Reformulation on Structured Data", In Proceeding of the ICDE, 2012.

[ZBWL+12] Junfeng Zhou, Zhifeng Bao, Wei Wang, Tok Wang Ling, Ziyang Chen, Xudong Lin, Jingfeng Guo, "Fast SLCA and ELCA Computation for XML Keyword Queries based on Set Intersection", In Proceedings of the ICDE conference, 2012.

[WDTY+12] Xiaoli Wang, Xiaofeng Ding, Anthony K.H. Tung, Shanshan Ying, Hai Jin, "An Efficient Graph Indexing Method", In Proceedings of the ICDE conference, 2012.

[WDTY+12] Xiaoli Wang, Xiaofeng Ding, Anthony K.H. Tung, Shanshan Ying, Hai Jin, "An Efficient Graph Indexing Method", In Proceedings of the ICDE conference, 2012.

[IBS08] Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman, "A Survey of Top-kQuery Processing Techniques in Relational Database Systems", ACM Computing Surveys,40(4):11, 2008.

[張13] 張祐愷,"利用ELCA技術對圖型資料庫搜尋R-clique", 碩士論文, 國立台灣海洋大學資訊工程研究所,2013.