以樹為基礎的點對點網路社群建構

黃文度 喬逸偉
佛光人文社會學院資訊學系
ywchiao@mail.fgu.edu.tw

摘要

這篇論文提出一個在點對點網路上利用樹的結構建構網路社群的分散式演算法，使得點對點網路內的節點可以根據其本身提出的專業興趣屬性自動組成一個緊密的社群網路(community networks)。而為了加速社群網路的形成，這個演算法依不同的興趣屬性為索引另外建立了一個參考樹，當新節點連結上點對點網路的時候就可以利用這個參考樹裡的資訊快速的找到自己有興趣的社群。因為整個結構是以樹為基礎，所以可以達到穩定的效能與擴充性。

關鍵詞：社群網路，點對點網路，樹

1. 前言

目前專業社群的意見交流與經驗分享除了定期舉行的學術會議與專業期刊之外，在網際網路上可以透過訂閱特定的新聞群組（news groups）或加入通信論壇（mailing lists）來達成。這樣的機制對於相關議題的專業討論是個方便的管道，但對於臨時的，突發的問題處理卻不見得是個知識分享的有效方式。舉例而言，有興趣的人員必須要先知道有這些新聞群組或通訊論壇的存在才能參與，而這些就需要搜尋引擎的協助。

在網際網路上搜尋開放分享的技術文件或知識資訊的方式主要仰賴在一般化的搜尋引擎如Google[6]，或專業社群網站，如 Citeseer[4]，的協助。但是搜尋引擎是利用關鍵字在數以億計的網頁中作幾近盲目的搜尋與索引，其結果不見得是使用者所期待或預期的。舉例而言，我們在 2005 年 8 月將 python 這個字丟進 Google 搜尋引擎裡查詢，結果傳回了約 28,400,000 個網頁。而其中至少包括了 Python 程式語言，某個著名英國劇團與某種特定爬蟲類生物等主題的相關網站與資訊。明顯的，並不是每個人都會對這三個差異懸殊的領域同時感到興趣，更別說那些可能埋藏在這些網頁之下而我們沒有注意到的其它可能主題。而在另一方面，如 Citeseer 這類的專業社群網站則不見得普遍存在於所有可能的專業社群之內。比如說，如果有一位家庭主婦想要上網搜尋或分享有關寵物蟒的照顧資訊。

為改善上述的這些問題，在網際網路上的相關研究已有語意網（semantic web）的努力[1]。主要的想法是為網頁加上特定的語義標籤與推演規則，之後就可以利用智慧代理人（intelligent agents）[10]在搜尋的過程中先進行語義的辨別與資訊的篩選。在這篇論文裡我們則著重在另一個方向：如果能夠將網路上有相同興趣的節點整合成一個社群網路，那麼知識的分享與討論應該可以更有效率而深入的進行。

這篇論文著重在網路社群的分化，特別是點對點網路（peer-to-peer networks）上的網路社群的建構。基於點對點網路開放，平等與分享的特性，它是目前相當重要的一個研究與應用領域。點對點網路利用網際網路開放的特性將網路上各個不同的獨立運算節點連結在一起，進行資訊的分享或知識的交流。除了目前最熱門的檔案分享應用，如BitTorrent[5]，之外，也有些應用是將整個點對點網路當成是一部鬆散連結（loss-coupled）的平行處理電腦進行繁雜的運算，如分析外太空無線電訊號的Seti@home 計畫[9]，或在點對點網路上建立分散式資料庫[3]，分散式檔案系統[8]等。

我們在這篇論文裡提出了一個將點對點網路依其興趣領域的不同自動分化成不同的次級網路的分散式演算法，以協助網路社群的形成。而為了加速社群網路的形成同時去除掉集中式索引節點的作用，這個演算法會利用興趣屬性作為索引建立一個參考樹。當新的節點連結上點對點網路但是沒有找到和自己興趣相符的網路社群時就可以利用這個參考樹的資訊快速的與相符的社群取得聯繫。

這篇論文的其餘部份安排如下：第二節介紹相關研究，第三節討論我們的演算法。第四節呈現演算法模擬的結果，同時整篇論文的貢獻總結在第五節。

2. 相關工作

[7]在 2002 年提出點對點網路社群的概念並同時提出了一個基於一種部份交集的定義方式。假設點對點網路裡的每一個節點都會宣稱許多種興趣屬性(attributes)，那麼我們可以定義點對點網路社群如下[7]：

- 點對點網路社群（p2p community）：一個非空的節點集合 N 稱為一個點對點網路社群若且唯若節點集合 N 有一個非空

（non-null）的簽名（signature）。

- 簽名（signature）：讓 n 代表點對點網路裡的一個節點，$claim(n)$代表節點 n 宣稱的興趣屬性集合，那麼節點集合 N 的簽名就定義為：$\bigcap_{k \in N} claim\ (k)$

由上面的定義可以看出，點對點網路社群的建構是以每個節點所宣稱的興趣屬性進行交集的計算，稱為這群節點的*簽名*（*signature*）。如果運算結果的簽名不是個空集合，那麼這群節點就可以稱為一個社群網路（community networks），以它們的簽名作為代表。更進一步，由這個定義來看，每個節點可以隸屬於不只一個社群，也可以藉由更改宣稱的興趣屬性而動態的改變它所隸屬的社群。

在社群的建構上([7]稱為*發現*)，[7]要求每個節點搜集它的鄰居節點與鄰居的鄰居節點宣稱的興趣屬性，並在有相同興趣屬性的節點間建立連結。當基於相同屬性的連結超過一個設定的門檻值，這個屬性就可以歸屬到社群簽名 S 內。而若發現有一群節點有共同的社群簽名 S，它們就屬於同一個社群網路。

SETS[2]則由搜尋的角度來考慮這個問題。如同在上一節提到的，如果能在進行網路搜尋時將搜尋的範圍縮小，那麼不但搜尋結果的精確度可以提高，搜尋的速度也可以加快。基於這樣的動機，SETS將每個節點的分享文件進行特徵值的運算，然後再將所有文件的特徵值組合成一個特徵向量稱為 *site vector*。具有相近特徵向量的節點就組合成一個主題叢集（cluster）。為了搜尋的目的，每個叢集都有一個特定的中心（center）節點，而這些不同叢集的中心節點就構成了整個點對點網路的*質心*（centroid）。質心的組成資訊是整個點對點網路都知道的開放資訊。新的節點加入網路時會先計算自己的特徵向量並與質心裡的特徵向量比對，然後加入有相近特徵向量的叢集。而資訊搜尋也利用類似的方式，透過質心將查詢傳送到相關的主題叢集內的節點進行。

這篇論文則提出一個以樹為基礎的點對點網路社群建構演算法。這個演算法主要的特點在：

- 分散式演算：去除領導節點或質心，不同社群的成員可以協助對方找到應屬的社群。
- 樹：社群網路的建構與維護是以樹的結構進行。而不是如[7]所採用的限制深度的 gossiping 的方式。

3. 演算法

在這一節裡我們說明這篇論文提出的網路社群建構演算法。首先是使用到的資料結構與演算法概觀。

3.1 演算法概觀

為了達到將點對點網路分群的目的，這個演算法將整個點對點網路視為一座樹林（forest），每個社群就構成這座樹林裡的一棵樹（tree）。因此整個社群的建構演算法也就可以分成兩個階段：首先找到這個節點在樹林裡應該屬於那一個群組(樹)，然後再將節點自身插入到那棵樹內相對應的位置，完成社群的建構。

為了支援前述的概念，每個節點p都和[7]裡的節點一樣，首先宣告自己有興趣的屬性(attributes)，以$claim_p$表示。和[7]不同的是，任意節點p在任何時刻只能宣稱一個興趣屬性，也就是說這個演算法裡每個節點在任何時間都僅能加入一個社群。更進一步，節點p維護了兩個列表，分別稱為$ALSC_p$（Address List of the Same Community）與$ALDC_p$（Address List of Different Communities）。其中$ALSC_p$記錄的是與節點p屬於同一個社群的節點位址，也就是說ALSC表記錄節點所屬的那棵樹的結構。當節點p接到一個有相同興趣屬性的節點q要求加入社群的訊息時，p就會啟動樹的節點插入演算法將q插入到適當的ALSC表內。ALSC表的結構如表 1 所示。其中第一筆資料欄位記錄父節點的位址。

表 1. ALSC 表

Key	Address
P	w.x.y.z
A	a.b.c.d

$ALDC_p$表則是將有不同興趣屬性的節點另外建構成樹結構，稱為*參考樹*。當節點p收到一個和它所屬的社群有不同的興趣屬性的節點r的加入要求時，節點p就利用這個$ALDC_p$內記錄的資料進行樹的搜尋將r的要求轉送到適當的群組節點。因此原來在$ALSC_p$表內的鍵值欄在這裡就成了叢集的屬性欄位：Cluster，如表 2 所示。和ALSC表相同，其中第一筆資料欄位記錄了這個節點在樹裡的父節點的位址。

表 2. ALDC 表

Cluster	Address
Sport	s.t.u.v
House Keeping	h.i.j.k

圖 1 顯示 ALDC 表的作用。在圖 1，宣稱興趣群組 B 的節點在加入點對點網路時首先接觸到的節點是興趣群組 A 裡的節點，利用 ALDC 表的資料進行了兩次訊息的交換之後透過群組 C 的節點找到正確的興趣群組。

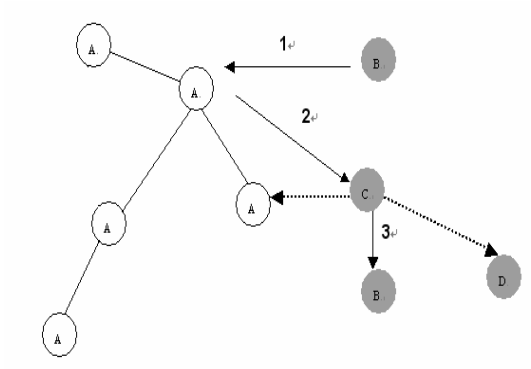

圖 1. 節點加入

do
GC_1:: (*isRoot*())
→ *broadcast*(***JOIN***, $claim_p$, *p*)

GC_2:: (*receive*():=(***JOIN***, $claim_q$, *q*))∧($claim_p$=$claim_q$)
→ *insert*($ALSC_p$, *q*)

GC_3:: (*receive*():=(***JOIN***, $claim_q$, *q*))∧($claim_p$≠$claim_q$)
→*forward*($ALDC_p$, *q*)

GC_4:: (*receive*():=(***Accept***, *q*))
→ *uplink*($ALSC_p$, *q*)
od

圖 2. 社群建構演算法

3.2 演算法

這篇論文的演算法如圖 2 所示。如前所述，每個節點*p*都維護兩個列表：$ALSC_p$與$ALDC_p$以及一個興趣屬性：$claim_p$。當節點*p*剛加入點對點網路時，$ALSC_p$與$ALDC_p$表都是空的，只有$claim_p$內記錄了節點*p*的興趣屬性。

演算法只允許樹的根節點可以發佈想要加入網路社群的訊息以避免節點不斷的重複加入。演算法一開始時，新的節點*p*本身形成一棵樹，它本身就是這棵樹的樹根。緊接著廣播節點*p*希望加入點對點網路社群的訊息，同時附上它的興趣屬性$claim_p$（演算法裡的GC_1）。而若節點*p*收到節點*q*希望加入的訊息，同時發現它們有相同的興趣屬性（$claim_p$=$claim_q$在GC_2），那麼節點*p*便會啟動樹的插入程序（GC_2的*insert*()），將節點*q*的資料加入由ALSC表所構成的分散樹中，最後由*q*的直接父節點*r*傳送一個***Accept***的訊息給節點*q*通知它它已經加入了相關的網路社群。節點*q*，當收到節點*r*的***Accept***訊息之後，會在$ALSC_q$中加入節點*r*的資訊並將父節點指向節點*r*（演算法裡的GC_4）。如果發生衝突的情形，也就是節點*p*，*q*互相都想要加入對方的樹時則可以藉由比較節點*p*，*q*的識別字（如*ip*位址）來處理。

另一方面，如果節點*p*收到與自己目前興趣屬性不同的節點*q*的加入要求，節點*p*則會啟動將*q*的資訊插入到由ALDC表所組成的參考樹之中的*forward*()程序（演算法裡的GC_3）。如果*q*的興趣屬性不是第一次在這個社群建構過程中出現，那麼在ALDC表的旅行過程中就會找到和節點*q*有相同興趣屬性的節點*r*，然後啟動將*q*插入有相同興趣屬性的ALSC表*insert*()程序（GC_2）。

3.2.1 insert()

演算法裡的 *insert*()是 ALSC 表建構的主要程序。執行 *insert* 程序的節點 *p* 可分為兩種角色：

- 根節點：節點 *p* 啟動樹的搜尋演算法，如果 *q* 應該存在 *p* 的子樹，*p* 就將 *q* 的 ***Join*** 訊息傳送給對應的子樹。
- 內部節點：如果樹的內部節點 *p* 由它的父節點（parents）接收到 *q* 的 ***Join*** 訊息，代表目前正在進行節點 *q* 的插入動作，節點 *p* 依照樹的插入演算法將 *q* 插入成自己的子節點（child），或進一步傳送給 *p* 的子樹處理。而若內部節點 *p* 是由外部或自己的子節點接收到 *q* 的 ***Join*** 訊息，節點 *p* 就將 ***Join*** 訊息繼續向上傳送。

3.2.2 forward()

在GC_3裡執行的*forward*()程序基本類似*insert*()，不過是在ALDC表上執行。當節點*p*收到一個和自己有不同興趣屬性的節點*q*的***Join***訊息時，節點*p*首先檢查自己的ALDC表，如果是空的，而且節點*p*並不是它所屬的網路社群的根節點，節點*p*就將*q*的訊息按ALSC表的記錄傳送給它的父節點。而如果節點*p*是它的網路社群的根節點，節點*p*就會啟動如同*insert*()程序的樹的插入演算法，不過這次是按照ALDC表的資料移動，將*q*的資料插入。而若在插入的過程中發現*q*的興趣屬性資料已經存在，如某個ALDC表內的節點*r*。節點*q*的***Join***訊息就會被轉送給*r*，啟動在ALSC表上的插入程序。

3.3 錯誤處理

上述的演算法有一個直接的問題是如果有節點離開網路，樹的結構就有可能遭受破壞而形成很多棵不同的子樹。這個問題可以在*insert*程序中處理。基本概念是如果有節點離開網路，那麼最晚在進行新節點的插入程序時就會發現有相對應的節點已經沒有回應。新的節點這時可以先插入這棵孤立的子樹成為它的新樹根，然後再重複進行在點對點網路內建構社群網路的動作（演算法裡的GC_1）。如果能與其它有相同興趣屬性的子樹取得聯繫，就可以將自己插入其它的子樹內，從而完成社群網路（樹）的合併。

4. 系統模擬

為了驗証這篇論文提出的演算法的穩定性與效能，我們利用個人電腦進行了演算法的模擬。並進行了三個不同的測試，分別在下面介紹。

4.1 環境設定

為了簡化進行模擬的環境，我們採用同步（synchronous）的分散式系統模型。系統提供一個全域（global）的時鐘。在每個時間單位，每個節點都可能處於下列三個狀態之一：

- 斷線：代表此節點目前沒有連結到點對點網路上。
- 休眠：代表此節點有連結到點對點網路上，但是並沒有進行這個演算法的運算。這是用來模擬現實中每個節點的運算速度或反應時間的不同。
- 運算：代表此節點正在執行這個演算法的運算。

為了模擬點對點網路實際上高度動態的狀況，任意節點可能在任意時間加入或離開網路，加上不同節點間的運算能力，連線速度都有可能不同的情形，模擬程式會隨機的選擇節點讓它休眠一段時間。節點休眠的時機與休眠的長度都是隨機決定。

而為了進一步模擬實際網路運算的非同步狀況，在每個時間單位所有節點執行的順序都會以亂數重排。以避免可能的順序執行影響到模擬的結果。

在效能評估上，因為演算法目的為網路社群的建構，因此我們主要的關心點在於是網路社群形成的速率。我們測量不同的網路社群總數隨時間增加而增加的速率來評估這個演算法的效能。而為了比較的目的，我們的實驗除了針對這個演算法進行模擬之外也進行了隨機接觸的社群建構的對照模擬。隨機接觸的社群建構是讓系統內的每個節點隨機的與其它節點接觸並交換資訊，如果發現有相同興趣屬性的節點就加入形成一個社群。如果發現興趣屬性不同，就不作任何事情，隔一段時間再重複前面的過程。

下面我們介紹實驗的結果。

4.2 實驗結果

這篇論文以三個主要的實驗來驗證這個演算法的效能：第一個實驗是以靜態的網路，節點加入網路後就不會再離開，來驗証這個演算法和隨機接觸的社群建構之間的效能比較。第二個實驗則同樣為靜態的網路設定，目標則在檢驗不同的興趣屬性個數對於這個演算法的效能影響。第三個也是最後一個實驗則使用動態的網路環境，重覆第一個實驗所進行的比較。

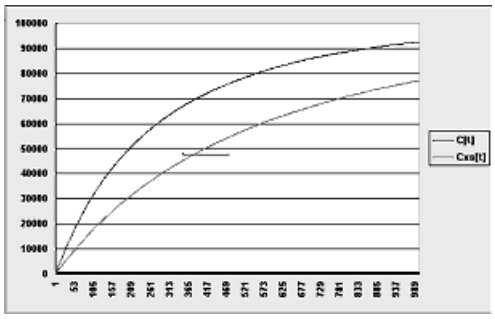

圖 3. 靜態點對點網路的社群建構

以下，我們分別介紹模擬的結果。

4.2.1 實驗一：靜態點對點網路的社群建構

在這個實驗裡我們主要是想知道在一個靜態的環境下，也就是節點加入網路就保持連線，不會任意離開，我們的社群建構演算法和純粹隨機接觸的方式之間的效能差異。

實驗條件

節點個數 100000，興趣屬性分為 1000 種，所有的節點都保持上線狀態。

實驗方式

我們在同樣的條件下，進行了兩次模擬：首先先讓所有的節點隨機的廣播它任意選定的興趣屬性，自然形成社群；第二次是使用我們的社群建構演算法。

實驗結果

模擬的結果如圖 3 所示。其中縱軸代表已找到相同興趣屬性形成叢集的節點數，而橫軸代表運行的時間。上方那條線代表使用我們演算法的結果，而下方的線則是純粹的隨機接觸的建構結果。

從圖 3 我們可以清楚的看出使用我們的建構演算法的社群連結速度明顯優於隨機接觸。我們進一步計算在不同時間點，兩者形成社群的節點數差異比率得到圖 4。

圖 4 的縱軸是以在同一時間點，兩次實驗形成叢集的節點數相減再除以隨機接觸方式的連結起來節點數所得到的比率。也就是說，如果我們以$C[t]$代表以我們的社群演算法在時點t已經加入叢集的節點數，$C_{xs}[t]$代表隨機接觸方式在時點t自然形成叢集的節點數，那麼圖 4 的縱軸表示的就是：$(C[t]-C_{xs}[t])/C_{xs}[t]*100\%$。

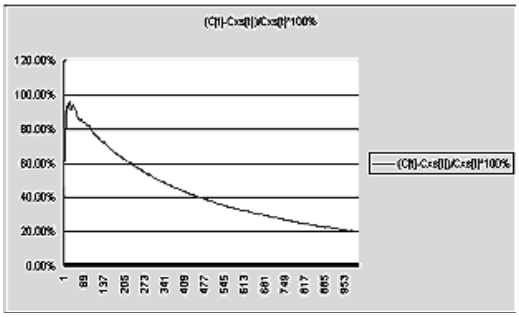

圖 4. 連線效率差異比較

由圖 4，我們可以看出我們的演算法在一開始的階段，也就是大多數的節點都還沒有形成叢集的時候有較佳的建構效率。而隨著叢集建構的大體完成，它的效率優勢也跟著緩步下滑。

4.2.2 實驗二：興趣屬性與社群建構在靜態點對點網路的關聯

在這個模擬我們想要知道同樣在一個靜態的點對點網路下，興趣屬性的個數與社群建構效率之間的關聯。

實驗條件

節點個數假定為 100000，興趣屬性種類（attributes）則設定在 100 到 10000 的範圍。所有的節點還是假設為靜態的，也就是說加入網路後就不會隨意離開。

實驗方式

為了解興趣屬性種類和社群建構之間的關係，我們將實驗分成兩組：第一組興趣屬性個數由 100 到 1000，每次增加 100 個興趣屬性，進行模擬；第二組則由 1000 到 10000，每次增加 1000 個興趣屬性進行模擬。

實驗結果

模擬的結果如圖 5 所示。橫軸代表時間，而縱軸則是在該時點已經加入叢集的節點數。

由圖 5 我們可以看見，如預期的，隨著興趣屬性個數的增加，也就是參與點對點網路的節點興趣愈分散，那它們之間形成叢集的速度就愈慢。這是因為每個節點在加入時，可以直接加入有相同興趣屬性叢集 ALSC 的機率變低的緣故，而大多需要透過 ALDC 的協助先找到和自己興趣相符的叢集才能加入，因而降低了叢集建構的效率。

4.2.3 實驗三：動態點對點網路的社群建構

前面兩個實驗的目的主要在了解這篇論文提出的社群建構演算法的特性，因此都假設了一個靜態的網路，所有節點都保持活躍（activated），而且不會離開。但現實的點對點網路則不是這個情況。相反的，現實的點對點網路是個高度動態的網路，也就是任意節點都可以在任意的時間斷線離開或加入網路。這個實驗就嘗試在這樣的設定下了解我們的演算法的表現。

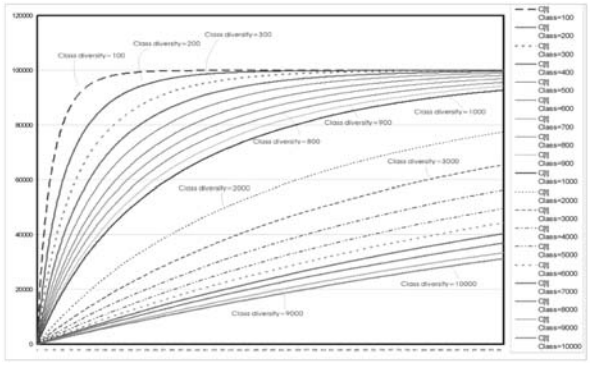

圖 5. 興趣屬性個數對社群建構的影響

實驗條件

節點個數的上限假定為 100000，興趣屬性種類（attributes）則設定在 1000，所有的節點都可能隨機的離開或加入網路。

實驗方式

起始條件設定為 3000 個活躍的節點，作為初始的網路，之後其它的節點可以隨機加入或離開。同樣以我們的演算法和隨機遭遇的方式進行模擬，並比較結果。

因為網路假設是動態的，所以統計的時候計算的節點數不是已形成叢集的節點數，而是在叢集內，且在統計時點仍然保持活躍的節點數。

實驗結果

模擬的結果如圖 6 所示。橫軸代表時間，縱軸代表在該時點已形成叢集而且仍保持活躍的節點數。其中上面那條線代表使用我們的演算法，下方那條線則是隨機接觸的方式。

由圖 6，可以看出我們的演算法在動態的網路下表現還是較隨機接觸的表現要好的多。但一個特別的地方在時點約 t=300 的地方有一個明顯的轉折。這個轉折的原因是出在我們實驗起始設定的 3000 個節點。因為模擬設定一開始就假定系統有 3000 個活躍的節點在線上，然後才開始動態的節點加入與離開。可能因為這個原因造成了在那個時點的轉折現象。

而我們設定系統啟始條件有 3000 個活躍節點的條件是為了簡化一開始網路社群建構的時間。因為每個節點在模擬中都只會在網路上延續一段任意的時間之後就會斷線。因此若讓所有的節點都在模擬開始的時候隨意的加入或離開網路，有可能造

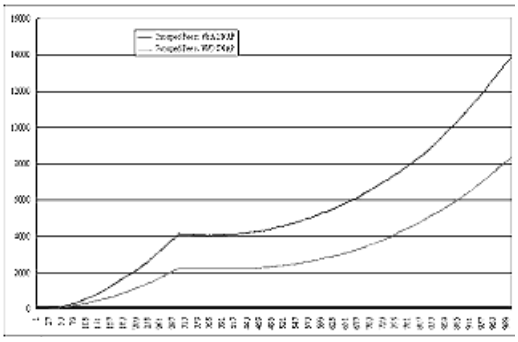

圖 6. 動態網路的社群建構

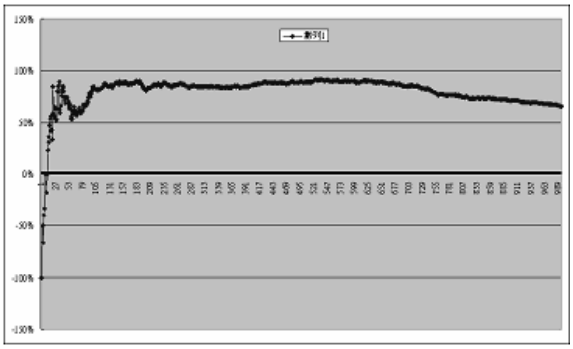

圖 7. 連線效率差異比較

成前面相當的一段時間都沒有社群網路的形成而只看到節點不斷的加入離開的動態過程。另一方面來說，設定 3000 個起始的活躍節點的則可以模擬一個基本的，尚未形成社群的點對點網路。然後我們的社群建構演算法才開始在這個網路基礎上開始運行。

更進一步，和第一個實驗相同，我們也進行了我們的演算法與隨機接觸的執行效能差異的比較。比較結果如圖 7 所示。

在圖 7 我們可以看到在動態的網路環境下，我們的演算法與隨機接觸的方式可以維持一個相當穩定的效能比，從而驗証這篇論文提出的演算法的穩定性。

5. 結論

在這篇論文我們提出了一個基於樹（tree）結構的點對點網路社群建構演算法。和其它的類似工作比較，這個演算法：

- 是完全分散式的。並不仰賴所謂的領導節點或質心（centroids）。
- 和隨機接觸的社群建構方式比較，我們的演算法顯示出穩定的效能優勢。
- 以樹為基本架構，因此在網路規模變大或社群個數增加時，效能不會大幅而快速的下降。

在進一步的工作方面，這個演算法是採用樹的結構，系統模擬時則是採用 B-Tree 的結構進行。但考慮實際的網路運作，那一種樹的結構是比較容易實現並提供較好的效率可能需要更進一步的研究探討。同時，這個演算法並不允許節點在同一時間宣稱多種興趣屬性而同時隸屬於多個網路社群。如何放寬這個限制，是另一個可能的進行方向。

最後，我們要謝謝論文審查者所提供的建設性意見，協助我們更進一步思考這份研究的揭限與改進了論文的呈現。

參考文獻

[1] G. Antoniou and F. van Harmelen, A Semantic Web Primer, The MIT Press, 2004

[2] M. Bawa, G. S. Manku, and P. Raghavan, “SETS: Search Enhanced by Topic Segmentation”, Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp.306-313, 2003.

[3] P. A. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini and I. Zaihrayeu, “Data Management for Peer-to-Peer Computing: A Vision”, In the Proceedings of the Workshop on the Web and Databases, 2002

[4] Citeseer, http://citeseer.ist.psu.edu/

[5] B. Cohen, “Incentives Build Robustness in BitTorrent”, In the Proceedings of the First Workshop on Economics of Peer-to-Peer Systems, 2003

[6] Google, http://www.google.com/

[7] M. Khambatti, K. D. Ryu, and P. Dasgupta, “Peer-to-peer Communities: Formation and Discovery.”, International Conference on Parallel and Distributed Computing Systems, pp. 161-166, 2002.

[8] A. Muthitacharoen, R. Morris, T. M. Gil and B. Chen, “Ivy: A Read/Write Peer-to-Peer File System”, Proceedings of the 5th USENIX Symposium on Operating Systems Design and Implementation, 2002

[9] Seti@home, http://setiathome.ssl.berkeley.edu/

[10] G. Weiss, Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence, The MIT Press, 2000