南　　華　　大　　學

資訊管理研究所

碩士論文

應用最大流量法於提升有向線性排列問題的下限解

Using the Maximum Flow Algorithm on Lower Bound of Directed Linear Arrangement Problem

研 究 生：曾雅君

指導教授：蔡德謙

中華民國 九十六 年 六 月

南　華　大　學

資訊管理學系

碩 士 學 位 論 文

應用最大流量法於提升有向線性排列問題的下限解

研究生：曾 雅 君

經考試合格特此證明

口試委員：

指導教授：

系主任(所長)：

口試日期：中華民國　96　年　6　月　8　日

誌　　謝

二年的研究生活，雖然生活不像大學那樣多彩多姿，但是卻得到很多收穫，不論是在專業領域上知識或是更圓融的人際關系，都讓我收益良多。

在此，首先感謝 蔡德謙老師二年來的耐心指導，在論文的研究期間給我許多正確的觀念及最佳的建議，謝謝 吳光閔主任及 吳金山老師對論文的批評與指教，學生會虛心接受，並將這些批評與指教蒙記在心。謝謝研究室的所有夥伴 小賴、珊珊、岳勳 以及可愛的學弟妹們，和大家在一起的時光，真的很開心。

最後，則感謝家人的默默支持，讓我在衝刺學業上無後顧之憂，以簡短的幾句話，表達對大家深深的謝意，謝謝!

應用最大流量法於提升有向線性排列問題的下限解

學生：曾雅君　　指導教授：蔡德謙 博士

南　華　大　學　資訊管理學系碩士班

摘　　要

隨著無線通訊技術的進步，行動通訊裝置已經普遍成為個人隨身配備，但由於無線的頻寬不像有線的頻寬那樣充裕，所以如何利用有效率的廣播來減少頻寬的使用和客戶端查詢時間，並提供最快速的搜尋效能便是在無線環境中重要的課題，因此本論文以無線廣播中廣播資料項之間存在的相依性為主要考量，使用有向無循環圖形（DAG）表示其順序限制，而每一資料項對應於圖形中每一頂點；並用最小路徑方法求得線性排列的最佳解；本論文以 Relabel-to-front 演算法求得 DAG 中，每個節點的最小分割(minimum cut)，再提出利用 joint vertex pair 提高 DAG 中的 Lower bound，最後則介紹如何運用此 Lower bound 進一步地降低 A*路徑搜尋法中，路徑搜尋的數量，並且尋找出一組最佳的拓撲排序。

透過模擬實驗結果的顯示，在本論文中所提出的方法確實可以縮短資料項之間的相依性長度，降低用戶端在收取所需資料項時花費的平均查詢讀取時間。

關鍵字： Relabel-to-front、Lower bound、A*搜尋法、minimum cut

Using the Maximum Flow Algorithm on Lower Bound of Directed Linear Arrangement Problem

Student：Ya Chun, Tseng Advisors：Dr. Derchian Tsaih.

Department of Information Management
The M.I.M. Program
Nan-Hua University

ABSTRACT

With the progress of wireless communication technology, the mobile device has been a necessary personal equipment. Since the bandwidth is not abundant compared to the wired environment, it is a important subject on making efficient broadcast to reduce the waiting time of user in the wireless environment.

Our context consider the existence of data dependence between each data item which are broadcasted. The directed acyclic graph was used with its vertex representing its data item and its edges representing the dependence between data item. The problem of optimal directed ordering is then can be found through the shortest path problem with the embedding graph.

In the context, we use the Relabel-to-front algorithm to find the minimum cut before each vertex in any topological order of DAG and also present a method which utilizing jointed vertex pair to increase the lower bound of cut before each vertex in any topological order of DAG. At last we introduce the algorithm which adopt this lower bound to prune the unnecessary path in the A* algorithm, which is used to find the optimal solution.

We provide a lower bound based on well known max-flow-min-cut theorem for A* algorithm .The performance results were shown and compared through extensive simulations. This broadcasted order of data item will then minimize the average response time for client's queries.

Keyword: Relabel-to-front、Lower bound、A* Search、minimum cut

目　　錄

表 目 錄

圖 目 錄

第一章 緒論

在有線的網路環境中，使用者需使用網路線才能與網路連接，這使得必需遊走各地的商務人事感到不便；而隨著科技的快速推進，成功的發展出無線網路（Wireless LAN 簡稱為WLAN）。一般來說無線網路的主要架構可以分為兩個部份：一為有基礎架構的無線網路（Infrastructure WLAN）；另一為無基礎架構的無線網路（Non infrastructure WLAN）；而本文討論的議題著重在有基礎架構的無線網路。

在有基礎架構下的無線網路中，行動通訊使用者利用其行動裝置(NB、PDA……等)所具備的無線網路連線功能，經由無線網路與基地台(Base Station, BS)或存取點(Access Point, AP)進行無線資料傳輸或通訊，再由基地台透過有線網路傳送到固定式主機；行動用戶端可以在不受空間及時間的限制下和世界通訊，如圖1。在有基礎架構的無線網路中，資料的廣播排程主要可分為推式(Push)[1,3,23]和拉式(Pull)[1,11,21,23,26]。

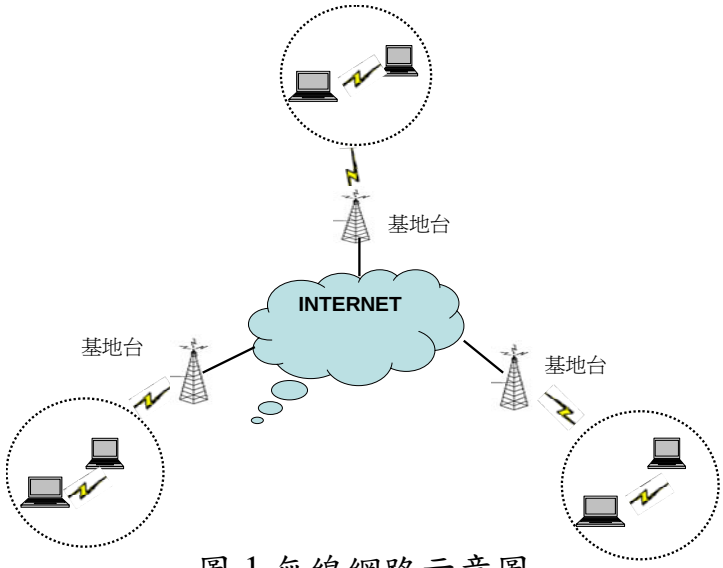


圖 1 無線網路示意圖

在push–based中，資料項在無線頻道上廣播，使用者只有接收的能力，此種廣播方式下，伺服器端會不斷地廣播資料，行動客戶端則是進入頻道持續聆聽伺服器端所廣播的資料，直到行動客戶端需要的資料項被播放出來，才將資料項截取下來。行動客戶端在這種廣播方式下，只能被動地等待與接收伺服器端廣播的資料，因此又稱為Data Broadcasting；如圖2。

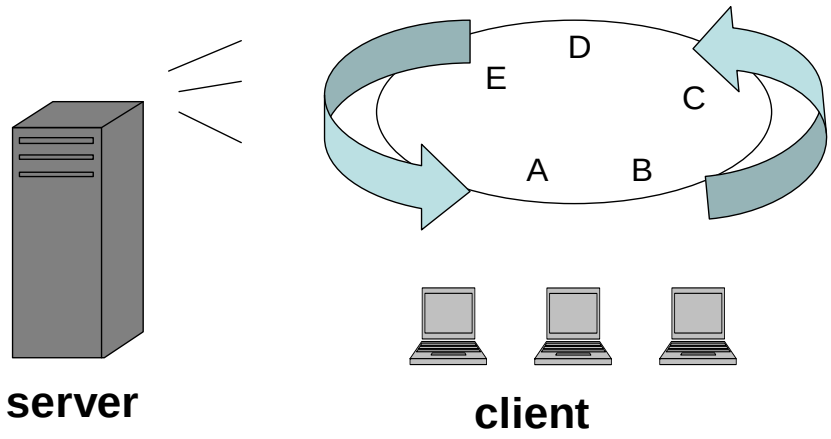


圖 2 Push–based 的廣播方式

而pull-based運作方式，行動客戶端同時具有接收和傳送的能力；首先，行動客戶端透過無線網路提供的上傳頻道(uplink channel)，讓行動客戶端可以利用上傳頻道向伺服器端提出請求，當伺服器端接收到行動客戶端的需求後，將透過單獨的下載頻道(download channel)傳送資料給行動客戶端；行動客戶端在這種方式下，可主動地向伺服器端提出請求。此方式又稱為需求模式On-Demand；如圖3。

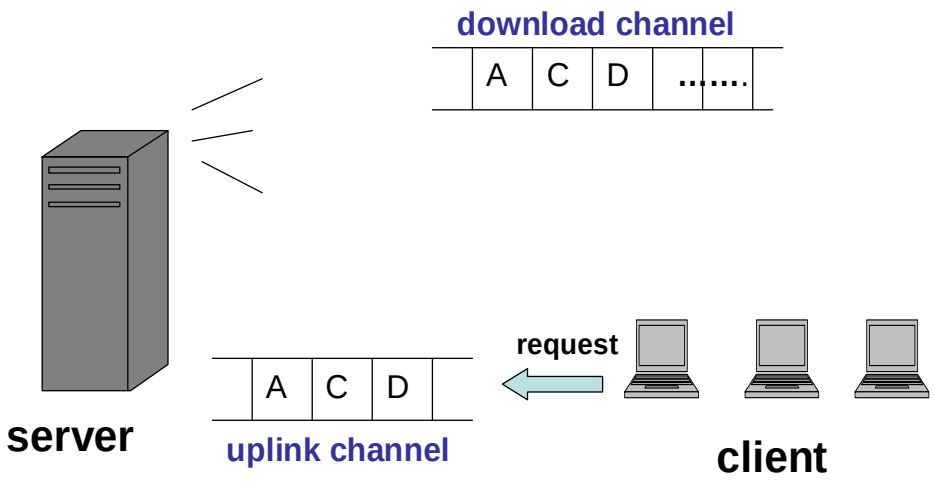


圖 3 pull-based 的廣播方式

1.1 研究背景與動機：

圖形理論起源於 1736 年，一位瑞士數學家尤拉(Euler)所提出；近年來，圖形的觀念廣泛地應用在資料結構中，如拓撲排序、最短路徑搜尋……等；圖形演算法是一種較為簡單明瞭的演算法，不僅涵蓋的範圍很廣，內容也很豐富；學術方面而言，已有很大的發展空間和研究價值。平時在日常生活中，我們經常也能找到與圖形演算法息息相關的內容。本論文，利用圖形理論來解決資料在廣播序列上的排序問題，以降低用戶端的平均查詢讀取時間。

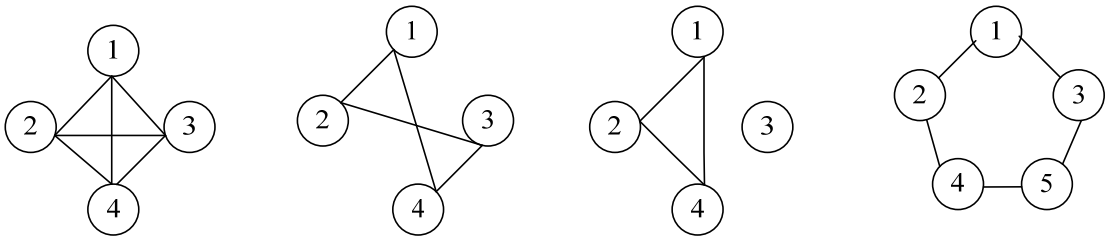
在圖形結構中[2,4]，圖形(graph)是由頂點(vertices)和邊(edges)所組成，以 G=(V, E)來表示；其中 V 為所有頂點的集合，E 為所有邊的集合；假設成對頂點之間的邊是不具方向性的，則圖形 G 為無向圖形(undirected graph);若成對頂點之間的邊是具有方向性的，則圖形 G 稱為有向圖形(directed graph)。

圖 4 中，顯示幾種不同類型之無向圖形。在一個無向圖形中，若存在一個由此頂點到另一個頂點的邊，即可稱為二頂點相鄰(adjacent)，如圖 2(c)，頂點 1 與 2 相鄰，頂點 1 與 4 相鄰，頂點 2 與

4 也相鄰。

具有 n 個頂點的無方向圖形，若存在“n(n-1)/2”條邊，則稱此無方向圖形為完整圖形(Complete graph)，如圖 4(a)。若每個成對頂點(V_i,V_j)都有路徑由V_i通到V_j, 則稱圖形是連通 (connected)圖形，反之；圖 4(c)中的頂點 3 沒有路徑至頂點 1、2、3，為不連通圖形。

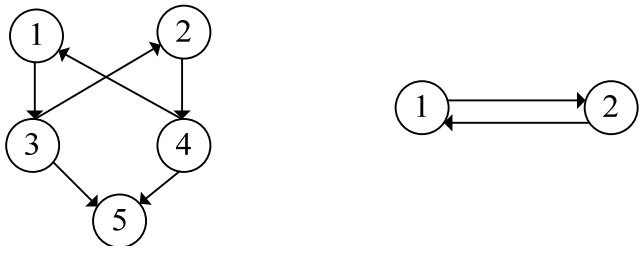
如圖 4(b)，假設起始頂點和終止頂點皆為同一個節點的路徑，稱之為循環路徑，如圖中 4(d){(1,2), (2,4), (4, 5), (5,3), (3,1)}，起點和終點都為頂點 1，因此為一個循環路徑。



(a)完整圖形　(b)連通圖形　(c)不連通圖形　(d)循環

圖 4 各種無向圖形

若頂點與頂點間的邊具有方向性，稱之為有向圖形；而假設此條有向路徑構成了一個循環，則稱為有向循環(directed cycle)，如圖 5(a)。而在有向圖形中，兩頂點間有兩條方向相反的邊, 稱之為強連通，如圖 5(b)。



(a)有向循環　(b)強連通

圖 5 各種有向圖形

Activity On Vertex Network(AOV 網路)為一種有向圖形,在 AOV 中,將一節點看作是一項必須執行的工作,其中有些動作沒有強制的先後關係,也就是可同時進行;而有些工作則有明確的先後順序關係。在頂點和頂點間具有方向性的邊,則代表每項工作與工作之間存在的先後關係順序限制。而本文利用的圖形結構-有向無循環圖(DAG)也屬於 AOV 網路的一種;在 DAG 圖中,以 G=(V,E)表示之,V=$\{v_1, v_2 \ldots\ldots v_n\}$為圖中之頂點集合,每一節點皆代表為一個可實行的工作;而 E 為所有邊之集合,$e_{ij} = (v_i, v_j) \in E$ 代表節點 i 的工作須先完成,才能進行頂點 j 的工作;近年來,在異質型(heterogeneous)系統中,DAG 排程相關研究[13,24]大致上分成 List scheduling 和 Level sorting 兩大方向:

List scheduling: 則根據每一個資料項之間的相依性建立其優先順序,再依優先順序將其排入廣播排程中,而每個資料項在排入廣播排程之前都必須先確定與排程中的資料項存在相依性才得以排入。

Level sorting:則將 DAG 中的節點集點分割成幾個部分,再針對每個子集合節點的作排序,試著將每個分割部分最佳化。

首先 level 0 包含所有節點v_j,但不存在任何節點v_i,$e_{ij} \in E$,即沒有先行者;Level k 包含所有節點v_j, 而v_i皆位於 k 的上層,$e_{ij} \in E$, 而且至少會有一個節點位於 k-1 層。如圖 6。

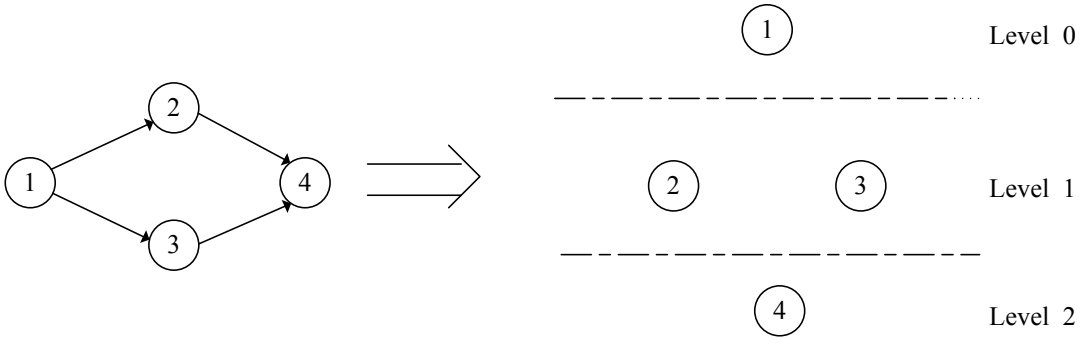


圖 6 DAG 轉 Level sorting

1.2 研究目的

在無線廣播中主要的議題著重於如何節省用戶端在接收資料時所耗費的電力問題[10,15,25]; 另一部份則是伺服器端要如何決定出一組較佳的廣播序列，縮短用戶端的等待時間。

無線網路中的通訊設備，如NB、PDA……等，均需電池的供給，因此電力節省的問題是相當受到討論的，如果行動客戶端在廣播頻道中的時間越長，就會增加電力的耗損；在過去研究中，有學者提出Selective tuning，利用廣播索引（indexing）[5,6,7,12,14,20,22,27]的方式降低行動用戶端在頻道中的時間；索引主要的功能是告知行動客戶端需要的資料項會在何時播放，當行動客戶端取得索引後，就不必在頻道中持續等待，可暫時先離開頻道，等到目標資料項播放時，再進入頻道擷取即可，因此tuning time大為減少，而電力的消耗也節省了許多。

另一個在無線廣播上被討論的議題則是降低用戶端的查詢時間，根據存取頻率將資料分為冷門資料和熱門資料，在廣播頻道中，提高熱門資料的廣播頻率來降低用戶端的等待時間。

在本論文中，著重在資料廣播排程的議題，利用有向無循環圖形(DAG)表示資料項所存在的相依性，利用 Relabel-to-front [28]，加上本論文提出 joint vertex pair 試圖提升 DAG 中的 Lower bound，以減少 A*路徑搜尋法中，開啟列表內路徑暫存數量。因而可以快速尋找出一組最佳的拓撲排序，而透過不斷的移動資料項在廣播序列上的位置，以求能降低用戶端的平均查詢讀取時間[8,17,18,19]。

1.3 研究架構

本章節主要在說明本研究的架構與概略地敘述各章節的目的：

第一章 序論

說明無線網路的背景、資料結構中圖形結構部分、本論文研究的動機與目的，以及對於本論文所期望達到的目標。

第二章 相關研究

彙整與本研究有關之文獻，包含、Relabel-to-front 演算法、拓撲排序以及路徑搜尋演算法(如 A*)。

第三章 問題描述

定義在本研究中所期望要解決的問題。

第四章 研究方法

詳細說明本論文應用圖形(DAG)以及 Relabel-to-front、Zero In-degree，A*路徑搜尋演算法，來解決廣播排程問題，縮短客戶端的等待時間。

第五章 實驗分析

決定各項參數並以 JAVA 程式語言撰寫程式進行模擬，再針對本研究在問題解決上所使用的演算法其效能分析。

第六章 結論

根據第五章進行模擬實驗得到的結果分析、比較。

第二章 相關研究

2.1 遞移封閉性(Transitive Closure)

所謂的遞移封閉性(Transitive Closure)，就是建立一條路徑矩陣(path matrix)為 P，從這個矩陣中可以瞭解圖形 G 中的任意兩個頂點間 i 及 j 是否存在一條從 i 可到達 j 的路徑。在此路徑矩陣 P 中只存在 0 與 1 兩種值，若 *P*[i][j]=0 表示 i 與 j 之間不存在任何一條路徑，若 *P*[i][j]=1 表示 i 與 j 這二頂點之間有一條路徑存在，且路徑長度大於等於 1。

建立路徑短陣的步驟為：

步驟一、分別求得 A^1，A^2，A^3，A^4….A^n，其中 n 為網路中的節點總數。

步驟二、將矩陣 A^1，A^2，A^3，A^4….A^n 相加，令 $P=A^1+A^2+A^3+A^4….+A^n$.

步驟三、令 P 矩陣中所有大於 0 的數字為 1，最後我們將得到只存在 0 與 1 的短陣，我們稱此矩陣為圖形 G，長度為 n 的 Transitive Closure。

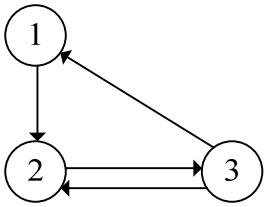


圖 7 Transitive Closure 範例圖

Example 2.1：在此我們例舉圖 7 說明 Transitive Closure 的演算過程；首先根據圖 7 先建立相鄰矩陣，在此相鄰矩陣中，1 代表頂點 i 和頂點 j 是相鄰的，反之設為 0。

步驟一: 根據圖 7，先求出相鄰矩陣 A^1

A^1=

	1	2	3
1	0	1	0
2	0	0	1
3	1	1	0

其中 $A^1[1][2]=1$ 表示頂點 1 到頂點 2 長度為 1 的路徑有 1 條。

步驟二: $A^2= A^1 A^1$

	1	2	3
1	0	1	0
2	0	0	1
3	1	1	0

x

	1	2	3
1	0	1	0
2	0	0	1
3	1	1	0

=

	1	2	3
1	0	0	1
2	1	1	0
3	0	1	1

步驟三: $A^3= A^2 A^1$

	1	2	3
1	0	0	1
2	1	1	0
3	0	1	1

x

	1	2	3
1	0	1	0
2	0	0	1
3	1	1	0

	1	2	3
1	1	1	0
2	0	1	1
3	1	1	1

步驟四: $P=\sum_{n=1}^{3} A^n[i][j]$

$$= \begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline 1 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 3 & 1 & 1 & 0 \end{array} + \begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline 1 & 0 & 0 & 1 \\ 2 & 1 & 1 & 0 \\ 3 & 0 & 1 & 1 \end{array} + \begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline 1 & 1 & 1 & 0 \\ 2 & 0 & 1 & 1 \\ 3 & 1 & 1 & 1 \end{array}$$

$$= \begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline 1 & 1 & 2 & 1 \\ 2 & 1 & 2 & 2 \\ 3 & 2 & 3 & 2 \end{array}$$

步驟五:最後令矩陣 P 中 P[i][j] >0 的項目都為 1。

	1	2	3
1	1	1	1
2	1	1	1
3	1	1	1

利用上述步驟即可求得其對應的 Transitive Closure。

2.2 拓樸序列(Topological Order)

在 AOV 網路中，假設頂點具有部份次序關係，可利用拓樸排序[16]可以將這些部份次序關係，轉換成線性次序(Linear Order)的關係。例如:頂點 i 是頂點 j 的前行者，在線性次序中，頂點 i 會排在頂點 j 之前的位置，具有這種特性的線性次序也可稱為拓樸序列(Topological Order)[1]。針對有向無循環圖形(DAG)，可利用深度優先搜尋法(Depth First Search)和 Zero In-Degree Sort 兩種方式轉成拓樸序列。

2.2.1 零分支度排序(Zero In-Degree Sort)

在 Zero In-Degree Sort 的方式中，首先尋找網路圖中任一節點的 in-degree 為 0(沒有先行者)，若同時有兩個以上的節點 in-degree 為 0，則任意選一，挑出此頂點後，並將與此頂點相連的所有邊刪除，不斷的重複上述兩步驟，直至所有節點皆被排序完成。重複上面的步驟，所產生的拓樸序列可能有一個以上，也就是所產生的拓樸序列並不是唯一解。

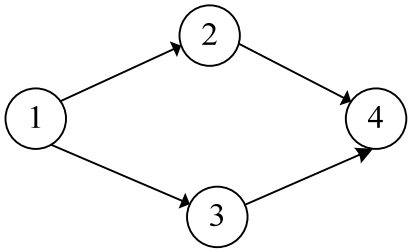


圖 8　Topological Order 範例圖

若以圖 8 進行 Zero In-Degree Sort 之拓撲排序如下:

1 找出所有 in-degree 為 0 的節點，在此圖形中只有節點 1

a.挑選出節點 1 放入序列中，並將其連接的邊皆刪去。

2.在剩下的節點中，找出 in-degree 為 0 的節點，此時有節點 2、3。

a.若有兩個以上 in-degree 為 0 的節點時，則任意選一頂點排入序列中。

b.在此假設挑選節點 2，並將其相連接之邊刪去。

3 在剩下的節點中，找出 in-degree 為 0 的節點，此時只有節點 3。

a. 挑選出節點 3 放入序列中，並將其連接的邊皆刪去。

4.最後，僅剩節點 4 還未放入序列中，因此將節點 4 也放入序列中。

5.得到其拓撲排列為:　1->2->3->4。

2.2.2 深度優先搜尋法(Depth First Search)

DFS 演算法是以堆疊 (Stack)來操作[30]。假設以某一頂點 v 為起點，接著找尋所有與 v 相鄰且未被走訪過的頂點中，選擇一個頂點 w 為新的起點。而 w 開始再度遞迴的方式執行深度優先搜尋法，若所有相鄰頂點皆拜訪過，則回溯到上一層繼續深度優先搜尋其他未被拜訪過的頂點，直到無法再找到未走訪過的相鄰頂點時。演算法如下：

若沿用圖例 8 說明深度優先搜尋法之拓樸排序：

1.假設以節點 1 為起點。

2.接著找尋所有與節點 1 相鄰且未被走訪過的頂點，在此假設挑選節點 3。

3.再找尋與節點 3 相鄰的節點 4

4.節點 4 已沒有相鄰頂點，因此，回溯到上一層繼續深度優先搜尋其他未被拜訪過的頂點，此時挑選節點 2。

5.最後得到其拓撲排列為: 1->3->4->2。

2.3 流量網路(Flow network)：

首先定義流量網路(flow network)[9]，$G=(V,E)$為一有向圖，其中 V 為所有頂點的集合，E 為所有邊的集合且每個邊容量皆為正值；若 $e_{uv}=(u,v)\in E$ 則定義為 $c(u,v)$，表示頂點 u 和頂點 v 間邊的容量(capacity)；而一個 Flow(流量)係一函數 f，以$|f|=\sum_{v\in V}(s,v)$表示，代表從 s 流至 t 的所有的流量值。S 表示 source(來源端)，而 T 表示 sink(目的地端)。

在網路流量中對任兩節點 u，v 而言，需滿足下列性質：

1.Capacity constraint: f(u,v)≤c(u,v)

2.Skew symmetric: f(u,v)=-f(v,u)

3.Flow conservation: 若 u∈V-{s,t}，則 $\sum_{v\in V} f(u,v) = 0$ 。

另外，由一 Flow network G 及一 Flow f 所導出的剩餘網路(Residual Networks) G_f ，也屬於 Flow network 的一種，其任兩頂點 u、v∈E 則容量定義為 C_f (u,v)=c(u,v)-f(u,v)表示頂點 u 和頂點 v 之間邊的剩餘容量。

對一個流量網路 G=(V,E)而言，Cut(S,T)將頂點集合 V 分割為 S 跟 T=V-S 兩部份且滿足 s∈S 及 t∈T；而通過 Cut(S,T)的網路流量定義為 $f(S,T) = \sum_{u\in S, v\in T} f(u,v)$；如圖 9，虛線切割圖形 G 為 Cut({s, v1, v2},{v3, v4, t})兩部分，計算通過 Cut(S,T)的網路流量為 12-4+11=19。

在流量網路中，假設已知一網路(每條邊上都有一流量值),且任一頂點 v，都存在一條從 source 到 sink 的路徑，試圖找出從 source(來源端)到 sink(目的地端)在此網路中的最大流量，便是在此議題中，研究的主要方向；以下小節將介紹幾種計算最大流量的演算法。

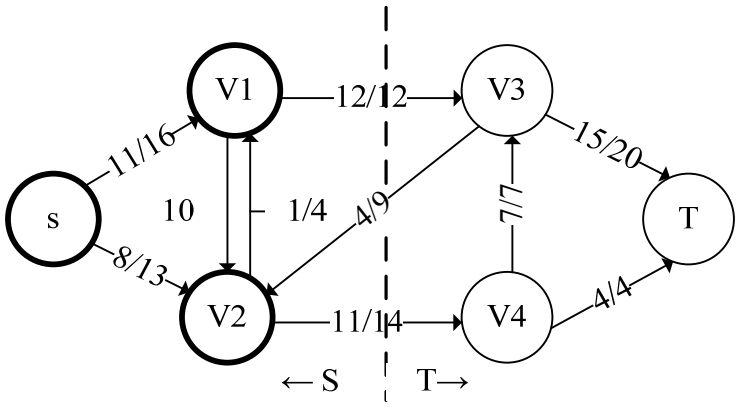


圖 9　Minimum cut

2.3.1 Ford-Fulkerson 演算法：

主要是利用 Residue network 的觀點重複地沿著增大路徑，來增大流量直到找到最大流量(Max flow)為止[9]。

演算法步驟如下：

1.找出 Augmenting path p。

2.將 Flow f 沿著 p 增加 $\min\{C_f(u,v)\}$ 且(u,v)在 p 上，即 residue network 在流動網路中路徑 p 上最小的容量(Capacity)

3.不斷地重覆 1.2 步驟，直到再也找不到一直從 source 到 sink 的增大路徑

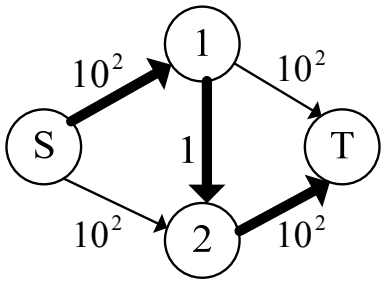
Ford-Fulkerson 演算法，雖然可以用來解決最大流量問題，但由於不能在線性時間內執行完成，所以並不是一個有效率的方法，利用下圖說明 Ford-Fulkerson 演算法。

Ex：圖 10 說明 Ford-Fulkerson 演算法，黑色粗線代表從 s 至 t 之增大路徑，虛線則代表增加的路徑。

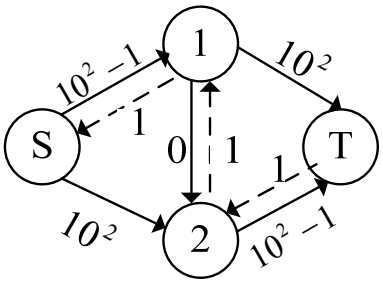
步驟一：黑色粗線為 s 至 t 的增大路徑，圖 10(a)的增大路徑只能傳送 1 單位的流量至目的地，圖 10(b)為執行第一次後的剩餘網路。

步驟二：黑色粗線為 s 至 t 的另一條增大路徑，圖 10(c)中，沿著增大路徑同樣傳送 1 單位的流量至目的地，而圖 10(d)為執行第二次後的剩餘網路。

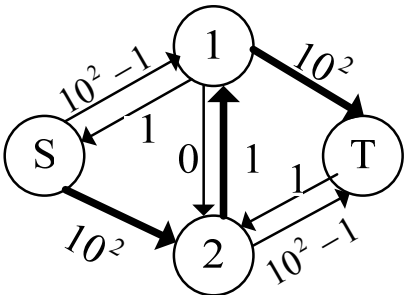
步驟三：黑色粗線為 s 至 t 的另一條增大路徑，圖 10(e)中，沿著增大路徑同樣傳送 1 單位的流量至目的地，而圖 10(f)為執行第三次後的剩餘網路。



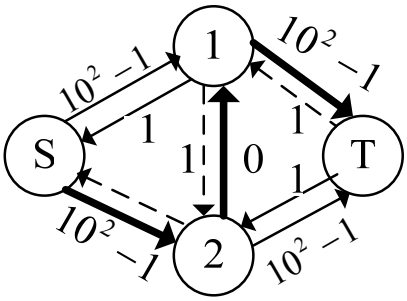
(a)原始網路圖，第一次找尋
另一條從 s 至 t 之增大路徑



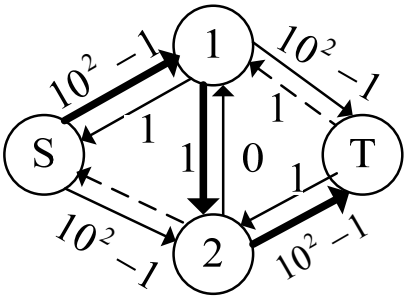
(b)執行第一次之剩餘網圖



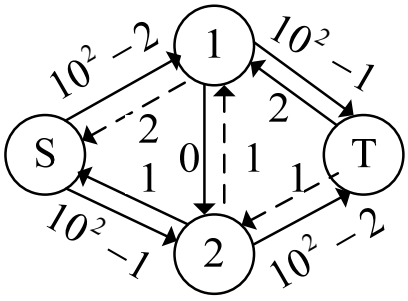
(c)第二次找尋另一條
從 s 至 t 之增大路徑



(d)執行第二次之剩餘網圖



(e)第三次找尋另一條
從 s 至 t 之增大路徑



(f)執行第三次之剩餘網圖

圖 10 執行 s 至 t 之增大路徑

由上面的圖例看來，Ford-Fulkerson 演算法執行至第三次時，開始重覆執行，所以至少需重覆執行 $2x10^2$ 次，才能將全部的流量送往

目的地端，因此效能並不好，所以有學者提出 Relabel-to-front 演算法來解決圖 10 的問題。

2.3.2 Relabel-to-front 演算法：

以下先介紹 Relabel-to-front 演算法[28]之符號定義及基本假設：

符號定義如下：

1.height(u)：節點 u 的高度。

2.excess(u)：頂點 u 目前存在的流量。

基本假設如下：

1. 預設 height(s) = |V|。|V|代表網路中，所有頂點的總數；而其它所有頂點 $x \in V$ 高度則設成 0，即 height(x) = 0。
2. 預設 excess(s)= ∞；其它所有頂點 $x \in V$，excess(x)=0。

假設欲找出 source(來源端)至 sink(目的地端)在此網路中的最大流量，除了可利用 Ford-Fulkerson 演算法之外，也可以使用 Relabel-to-front 演算法來找出網路中最大流量，此演算法包含三大部分，可藉由調整、改變每個頂點的高度，即可以將流量順利流到目的地；換句話說，在網路中的每個頂點(除了 s 和 t)都能適時改變高度，將這些流量在網路中傳送，直到將這些流量送至 sink(目的地端)為止，此演算法所需時間為 $O(V^3)$。

第一部分、push 演算法：

由頂點 u 傳送部分流量(即頂點 u 之存在流量與剩餘流量的最小值)至頂點 v。

首先必需滿足下列條件，才能執行 push 演算法，：

1.excess(u) > 0：頂點 u 目前存在的流量大於 0 。

2.c(u, v) - f(u, v)> 0：頂點 u 到頂點 v 間剩餘容量需大於 0 。

3.height(u) > height(v)： 頂點 u 的高度要比頂點 v 的高度高。

滿足上列條件後，則傳送 min[excess(u),c(u, v) - f(u, v)]的流量至相鄰節點 v。

演算步驟如下：

Algorithm 1 Push

```
Begin
Input： u, v ∈ V
1. send = min(excess[u], C[u][v] - F[u][v])
2. F[u][v] += send
3. F[v][u] -= send
4. excess[u] -= send
5. excess[v] += send
```

第二部分、Relabel 演算法：

若頂點 u 的高度低於頂點 v 的高度，不能成功的將流量 push 至頂點 v，因此執行 Relabel 演算法，使頂點 u 不斷地增加其高度，直到至少高於某一相鄰節點。

下列兩個條件為將頂點 u 之 flow 傳送到頂點 v 之必要條件：

1. excess(u) > 0：頂點 u 目前存在的流量大於 0。
2. $height(u) \le height(v)$：頂點 u 高度比其它所有頂點低。

演算步驟如下：

Algorithm 2 Relabel

```
Begin
Input ： u ∈ V
1.Store the height of the node u
2.For k=0 to n-1do
3.  If (c[u][v]-f[u][v]>0) then
4.    min_height = min(min_height, height[v])
5.    height[u] = min_height + 1
6.  End If
7.End For
```

第三部分、discharge 演算法：

在此演算法中，若還有流量存在某頂點 u，則對頂點 u 之所有相鄰頂點瞭解是否可執行過 Push 演算法；若不能執行 Push 演算法時，則立即執行 relabel 以提升該頂點高度。

演算法如下：

Algorithm 3 Dischange

```
Begin
Input： u∈ V
1. while excess(u) >0
2.  If not all neighbours have been tried then
3.    If (c[u][v]-f[u][v]>0 & height[u]>height[v])then
        push(u,v)
4.    End If
5.  relabel(u)
6.  End If
7.End While
```

介紹完 Relabel-to-front 所包含的三部分後，此演算法的詳細步驟

如下：

步驟一、將不是 source 和不是 sink 的頂點放至串列中。

步驟二、source node 將流量流至與其 source 直接相鄰的所有節點,且 source 流至這些節點的流量即路徑所能容納流量(c[u][v])。

步驟三、對串列中的節點試著去執行 push 演算法，若 height(u)>height(v)，則節點 u 可取 min(excess(u),c[u][v] - f[u][v])的流量至節點 v。

步驟四、任一節點 u∈V , excess(u)>0 但 height(u)<height(v)，則需執行 Relabel 演算法：不斷地將高度加 1，直到至少比相鄰的節點高度還高，就能順利的執行 push 的動作。

步驟五、經過上面幾個步驟後，頂點的高度若有改變的話，將此頂點移至串列的第一個位置，再從串列的第一個頂點重覆上述動作。

步驟六、不斷重複 Relabel 和 Push 演算法，直到再也不能將流量 push 至相鄰節點為止，此時從 source node 所流出的總流量，就是此網路圖的最大流量。

EX：

利用圖 11 之 DAG 圖詳細說明 Relabel-to-front 演算法的執行過程，以求得在此範例中 source(來源端)至 sink(目的地端)的最大流量(假設節點 0 為 source，節點 3 為 sink)。

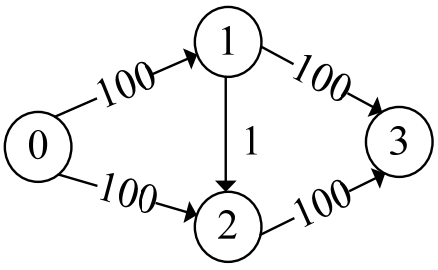


圖 11 Relabel-to-front 演算法之範例圖

步驟一：

i. 根據演算法的定義得知 height(0)=4, excess(0)=∞；其餘節點的高度和 excess 皆為 0。

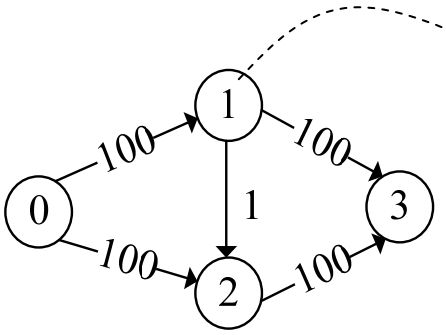
ii. 接著將不是 source 和不是 sink 的節點加入串列中(將節點 1、2 加入串列中)，

步驟二：

i. source 推送流量至鄰近節點;source 取 min(∞, 100)的流量至節點 1；而取 min(∞, 100)的流量至節點 2 。

ii. 因此，節點 1 目前的 excess(1)=100，節點 2 目前的 excess(2)=100。如圖 12(a)。

步驟三：節點 1 需 push 流量至 sink，但不符合 height(1)>height(3)的條件，因此執行 relabel 將高度增加為 1。如圖 12(a)。

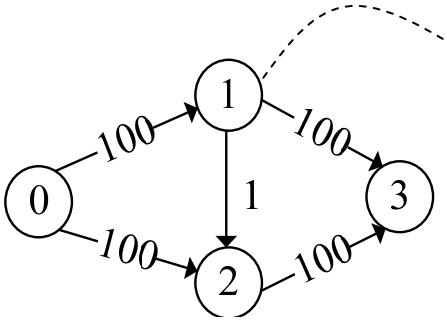


height	push	To node	excess
0	0		100
1	1	2	99
1	99	3	0

height	push	To node	excess
0	0		101
1	100	3	1
2	1	1	0

將剩下的 flow 倒回 node1

(a) 應用 Relabel to Front 於圖 11 範例之步驟一 ~ 步驟五



height	push	To node	excess
1			1
1	1	3	0

(b) 應用 Relabel to Front 於圖 11 範例之步驟六 ~ 步驟八

圖 12 應用 Relabel to Front 於圖 11 範例

步驟四：

i. 節點 1 的高度增加為 1 後，可推送流量至鄰近節點，取 min(100, 1)的流量 push 至節點 2。如圖 12(a)。

ii. 節點 1 的高度增加為 1 後，同樣地取 min(99,100)的流量 push 至 sink(節點 3)。如圖 12(a)。

步驟五：

i. 節點 2 接收節點 1 所 push 的流量，因此 excess(2)=101。如圖 12(a)。

ii. 此時節點 2，因不符合 height(2)>height(3) 的條件，所有執行 relabel，將高度增加為 1 後，取 min(101,100)的流量 push 至 sink(節點 3)。如圖 12(a)。

步驟六：

i. 節點 2 的 excess(2)=1 ,不符合 height(2)>height(1)的條件。

ii. 執行 relabel 將高度增加為 2，符合 height(2)>height(1)，最後將 excess(2)=1 倒回節點 1；如圖 12(a)。

步驟七：

i. 由於節點 2 倒回 1 單位的流量至節點 1，因此 excess(1)=1。如圖 12(b)。

ii. 符合 height(1)>height(3)，取 min(1,0)的流量 push 至 sink(節點 3)。如圖 12(b)。

步驟八：當 source 將所有 flow 流至 sink，即計算從 source node 所流出的總流量=100+100=200，計算總流量後，演算法停止運作。

利用相同的例子，比較 Ford-Fulkerson 演算法和 Relabel-to-front 演算法的執行次數，發現使用 Ford-Fulkerson 演算法需執行 2x100 次，才能將流量流至目的地端；而使用 Relabel-to-front 演算法，不需執行 200 次就可成功的將流量流至目的地。因此，Relabel-to-front 演算法有較好的執行效率。

2.4 圖形搜尋法

2.4.1 A*搜尋演算法

A*演算法是圖形搜尋中[29]，用來解決解決最短路徑(Shortest Path)問題的一種演算法，在此演算法中，利用啟發式估計（heuristic estimate， $h(x)$）來預測透過此節點 x 到目的地是否有最小的成本估計值，因此可以快速地將許多明顯為壞的路徑排除考慮, 進而計算出一條較佳的路徑到目的地，計算的公式如下：

G(x): 從來源端到目前節點的距離

H(x): 預測目前節點到終點的距離(此為 A* 演算法的主要估計公式)

F(x): 目前節點的估計值

F(x) = G(x) + H(x)　　公式一

A*搜尋演算法簡單地來說首先將來源端加入「Open list」中，接著重複如下的工作：

1. 利用公式一算出，來源端附近的節點的 F 值，選出 F 值最小的為下一個"傳送點"，並將此節點存入「Close list」中。

2.對"傳送點"相鄰的每個節點去做以下的判斷，如果它不在 Open list，把它添加進去，並把傳送點作為此節點的父節點，接著記錄這一格的 F, G 和 H 值。相對的如果它已經在 Open list 中，此時藉由 G 值的大小來判斷是否有更好的路徑(值愈小，代表路徑較好)，如果 G 值是比原本的小，那麼就設此"傳送點"為節點的父節點。

3.最後把目標格加入 Close list，代表找到路徑。

EX：

圖 13 為一 A*搜尋演算法範例圖。在圖中，S 為 Source，T 為 Sink，灰色部分為障礙物。G 代表從來源端到目前節點的距離；在這個例子裡，我們令水平或者垂直移動的代價為 10，對角線方向代價為 14，而 H(X) 則是預測目前節點到終點的距離所以我們只考慮水平和垂直的方格的數量總和，忽略對角線方向，而箭頭指向為父節點。在此圖例中，方格中的左下角的值代表 G(x)，右下角的值代表 H(x)，左

上角的值代長 F(x)。

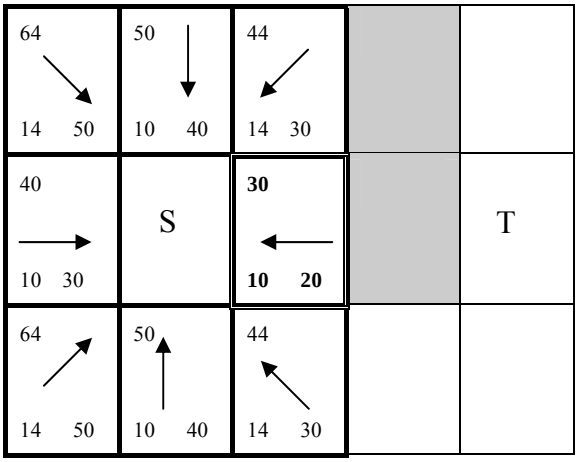


圖 13 A*搜尋演算法範例圖

步驟一、從 Source 附近的節點(框著粗線的部分，有 8 個方格)選出最小的 F(x)值，為 F(x)=30，先設此點為傳送點。如圖 14。

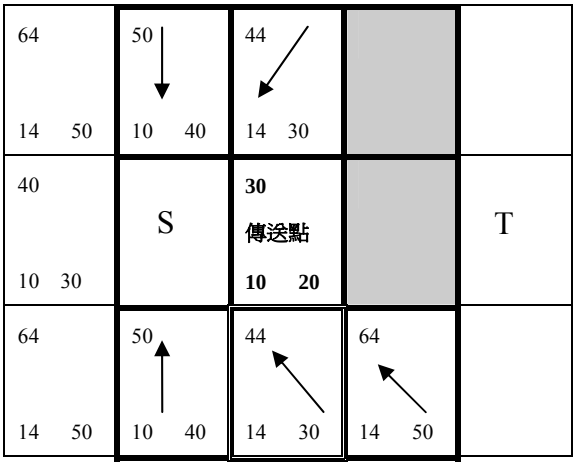


圖 14 應用 A*搜尋演算法於圖 13 範例步驟一

步驟二、計算此傳送點附近的節點的 F(x)，找到 F(x)=44，但是此節點如果透過圖中的傳送點再到 Source，G(x)值為 20，比原來的 G(x)大，所以不選擇通過 F(x)=30 的傳送點再到達 S，

而是選擇直接到達 S。如圖 15。

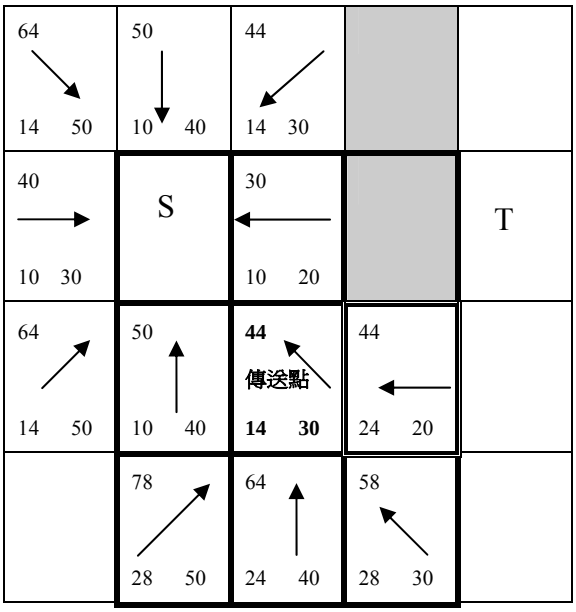


圖 15 應用 A*搜尋演算法於圖 13 範例步驟二

步驟三、計算此傳送點附近的節點的 F(x)，找到 F(x)=34，所以設此點為傳送點。最後，虛點曲線為 Source 到 Sink 的最佳路線。如圖 15。

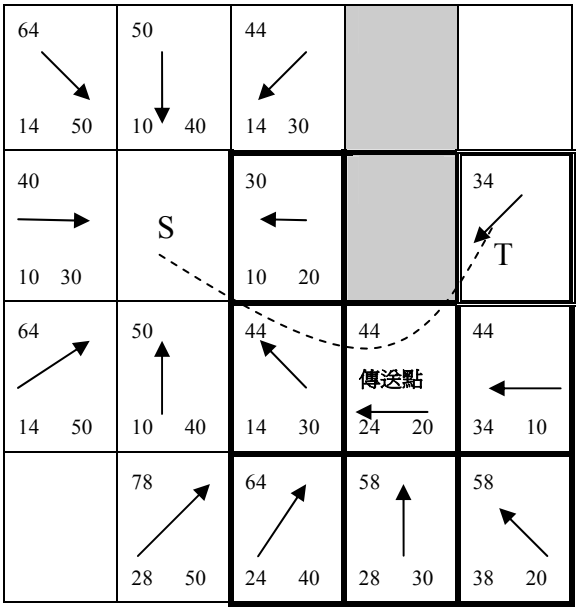


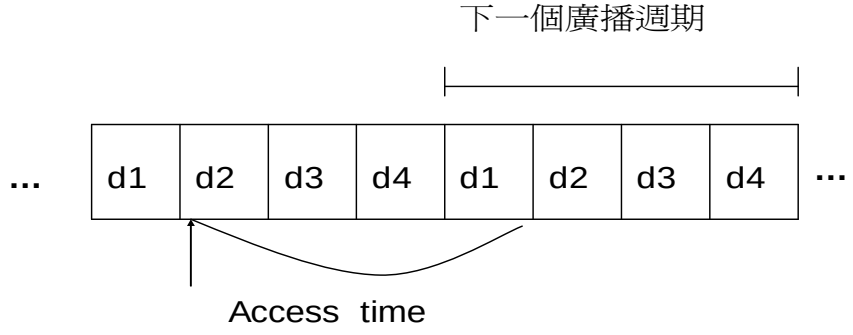
圖 16 應用 A*搜尋演算法於圖 13 範例步驟三

第三章 問題描述

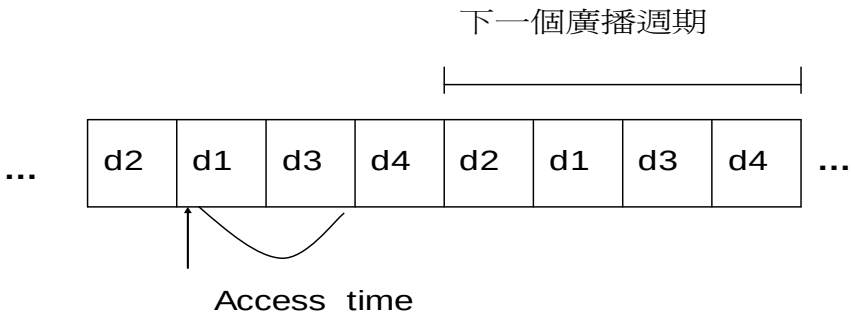
3.1 平均查詢讀取時間

在有基礎建設下的無線網路，大致上可分成 push 和 pull 兩種廣播方式，push 系統則是由伺服器週期性的廣播資料，行動用戶端如果聽到伺服器廣播的資料正是自己所需要的，便將資料擷取下來；而 pull 式系統，行動用戶端可主動地向伺服器端提出請求(Request)，伺服器收到請求後將行動用戶端所需的資料項直接傳給客戶端。然而在無線環境中，行動設備(NB、PDA)的電力是有限的，所以如何縮短客戶端擷取所需資料的時間是重要的課題之一[8,13,14]。

假設行動客戶端要收取的資料為 d1 及 d3，Access time(存取時間)如圖 17(a)所示；但如果將資料的廣播排程順序改變，如圖 17(b)，比較 17(a)、17(b)明顯地發現 Access time 縮短；因此，在廣播資料排程中，資料的排序對客戶端的平均查詢讀取時間有很大地影響。在本論文中，我們利用資料相依的特性，假設行動客戶端對於所請求的資料項是有收取先後順序的限制，再加上 Relabel-to-front 演算法以及 A* 搜尋演算法來解決資料廣播排程中存取時間的問題，降低客戶端在收取所需資料項時花費的平均查詢讀取時間。



(a)沒有根據資料相依性排列的廣播排程



(b)根據資料相依性排列的廣播排程

圖 17 廣播排程中的存取時間(Access time)

3.2 有向線性排列的最佳化問題

有向線性排列的最佳化問題(Directed Optimal Linear Arrangement)所要探討的就是有關資料廣播排程最佳化的問題，藉此降低用戶端的Access time；舉例來說，一個行動客戶端欲查詢 yahoo 的財經新聞，首先點選進入 yahoo 的首頁，接著查詢新聞，再查詢所要查的財經新聞，這表示行動客戶端對伺服器的查詢要求存在先後次序，相對地，行動用戶端對於所要求的資料項也存在收取先後順序限制。

在有向線性排列的最佳化問題中，以圖形 G=(V, E, w)表示之，資料項對映為有向圖形中的節點集合 V；成對資料項之間的相依性對映為有向圖形中邊的集合 E，兩資料項間的相依估計值則對映為有向圖形中，邊的權重值 w(此估計值可用其它方式求得，但不在本論文的討論範圍內)。

另外，在有向線性排列的最佳化問題中，假設圖形 G，存在一條從節點 u 到節點 v 的路徑，即(u,v)∈E，若依順序來排列，節點 v 出現在節點 u 之後⇒f(u) <f(v)；函數 f→{1,2,…|V|}，f 代表拓撲的順序。$W(f)$則代表在 f 排序下所有邊的總權重延遲，以下為公式：

$$W(f) = \sum_{(u,v)\in E} w_{u,v}(f(v) - f(u)) \qquad \text{公式二}$$

$w_{u,v}$為節點 u 到節點 v 間，邊的權重。

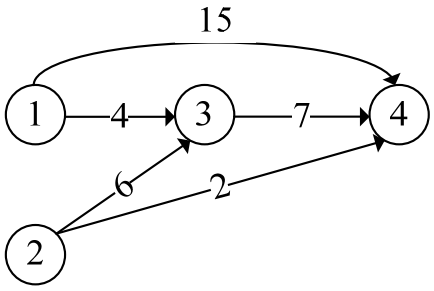
$f(u)$、$f(v)$：節點 u 和節點 v 在 f 排列中的相對位置。

$f(v)-f(u)$：節點 u 和節點 v 在 f 排列中的距離。

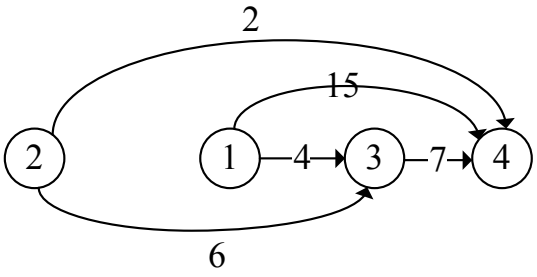
Ex：

例舉圖 18 說明上述之所有邊的最小成本值。圖 18(a)為一個有向無循環圖，圖 18(b)則是圖 a 的一個拓撲排序圖，f(1)= 2，f(2)=1，f(3)=3，f(4)=4。

代入公式一所算出的最佳線性規劃的最小成本值 $W(f)$=2x3+15x2+4+7+6x2=59。



(a)



(b)

圖 18　(a)一個有向的無循環圖　(b) 拓撲排序圖(TSG)

令表示 A 群組和 B 群組的分割權重，從公式中，給定一個排序 f，

$W_i(f)$代表切割介於第 $f(i)$和第 $f(i+1)$之間的分割權重(cut weight)，而公式二可以重寫如下：

$$W(f)=\sum_{i=1}^{n-1}W_i(f)$$

$$=\sum_{i=1}^{n-1}\sum_{u,v\in V \&\& f(u)\le i<f(v)}W_{u,v} \qquad \text{公式三}$$

Ex：

例舉圖 18，圖 18(b)則是圖 a 拓撲排序圖，f(1)= 2，f(2)=1，f(3)=3，f(4)=4。若分割權重(cut weight)介於第一個頂點和第二個頂點為$W_1(f)$=Scut({2},{1,3,4})，若分割權重(cut weight)介於第二個頂點和第三個頂點為$W_2(f)$= Scut({2,1}, {3,4})，依照上述的作法以此類推，$W(f)$ = (邊的總權重延遲)= $W_1(f)$ + $W_2(f)$ + $W_3(f)$ = (2+6)+(2+15+4+6)+(2+15+7)=59，此結果和代入公式一所算出的$W(f)$=2x3+15x2+4+7+6x2=59 相同。

第四章 研究方法

為了解決相依性資料項廣播排程問題。在這個部份中，將介紹提升 Lower bound 的方法並加入 Relabel- to-front 方式之拓樸排序應用於廣播排程問題的演算流程，並使用例子來加以詳細說明。

4.1 廣播查詢轉換有向圖形

下列為演算法中使用到的符號定義說明:

- ✓ V: 所有頂點集合。資料項對映為有向圖形中的節點集合 V。
- ✓ E: 連接二個頂點的有向邊(edge)集合。也就是成對資料項之間的相依性集合。
- ✓ w：兩資料項間的相依估計值，即有向圖形中邊上的權重值。(可用其它方式求得此估計值，但不在本論文的討論範圍內)。

本論文在有向無循環圖(DAG)的圖形架構下，利用 joint vertex pair 增加其 DAG 中的 Lower bound，再利用利用 A*搜尋法找出最佳的排序，以縮短行動客戶端的平均讀取時間。若遇到循環圖形時，則可將造成循環的有向邊(edge)減去最小值，以解決有向圖形中循環的問題。從接下來的 4.2 節到 4.5 節則詳細說明轉換成有向圖形後，如何利用所提出的演算法將資料項根據相依性大小排列，以縮短行動客戶端的平均讀取時間。

4.2 Lower bound from Max-Flow Min-Cut Theorem

此小節介紹 Lower bound 用以解決有關資料廣播排程最佳化問題。將任一頂點 u∈V 的出射權重(outgoing weight)由大到小依序排

列，定義成 $w_u^1 \geq w_u^2 \geq \ldots\ldots \geq w_u^d$，則 $X^+(u) = \sum i w_u^i$ 表示頂點 u 所有 outgoing weight 的總流量最小延遲加總；反之 $X^-(u)$ 則定義成，頂點 u 所有 incoming weight 的總流量最小延遲加總；從公式四看來，此方法僅考慮排序位置及每一頂點 outgoing weight 和 incoming weight 之權重，因此可有效率的提供下限解。

$$W(f) \geq \max(\sum_{u \in V} X^+(u), \sum_{u \in V} X^-(u)) \qquad \text{公式四}$$

但如果需要一個更佳的下限解，即可考慮藉由 Max-Flow Min-Cut Theorem 求得；首先定義 DAG 圖中的 V 集合切割為 V^l 以及 V^u（V^u = V \ V^l），分別稱之下層群組(The Lower Set)以及上層群組(Upper Vertex Set)，而 M(U)代表 U 集合的最小來源集合。

在 Max-Flow Min-Cut Theorem 中，任一頂點 u∈V，將頂點 u 之前的稱之 V^l（下群組），頂點 u 和其之後的稱為 V^u（上群組）；$Scut(V^l, V^u)$ 則表示 V^l 和 V^u 的分割權重；在此定義 $Mcut(u)$ 為在任何有效的（V^l，V^u）中，頂點 u 之前的權重切割(weight cut)為最小的，即 $u \in M(V^u)$。如公式五：

$$Mcut(u) = \min_{(V^l, V^u), u \in M(V^u)} Scut(V^l, V^u) \qquad \text{公式五}$$

對 $u \in M(V^u)$ 的所有節點都有一條路徑到頂點 u，將其放至 V^l；而從頂點 u 的 outgoing 相鄰之頂點，將其放至 V^u。從最小切割最大流量定律中，最小切割問題的解答可透過最大流量問題藉由將至少有一條路徑至 u 的所有頂點，皆稱為 source；而頂點 u 和與頂點 u 位於同一條路徑的 outgoing 頂點，皆稱為 sink；即假設 u 在 $M(V^u)$ 中，則在

(V^l, V^u) 中的最小切割等於 $\{v \mid v \prec u\}$ 和 $\{u\} \cup \{v \mid u \prec v\}$ 的最大流量。

在多來源及多目的地(multiple sources and multiple sinks)之網路中，藉由增加虛擬的super source和super sink至DAG圖中，即可與單一source/sink的作法相同；在此網路中super source有∞容量流至source端, 而且∞容量流出至super sink中，同樣地我們想找出在此網路中任一節點的最大流量，在本論文中，我們使用Mcut(i)來表示之(符號定義在第二章已介紹，在此不再贅述)。

假使欲計算任一節點 m∈V，節點 m 的最大流量(mini cut)。首先，利用遞移封閉性(Transitive Closure)的方法，找出哪些節點是在同一條路徑上。將 m 頂點之前且與 m 頂點在同一條路徑的節點相連至 super source，再將 m 頂點之後且與 m 頂點在同一條路徑的節點，包括自己本身(節點 m)都相連至 super sink；最後則是將不在同一條路徑上的節點加至串列中。

串列中的節點則利用第二章文獻探討中提到的 Relabel-to-front 演算法，不斷地增加本身的高度，對相鄰節點執行 push 的演算法，試著把流量推向目的地端；最後再將 source 直接相連至 sink 的流量加上使用 Relabel-to-front 演算法算出的流量加總，就可以得知此節點在網路中的最大流量(即最小切割)。

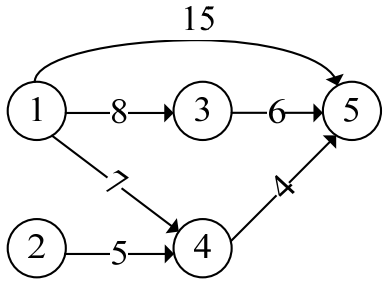
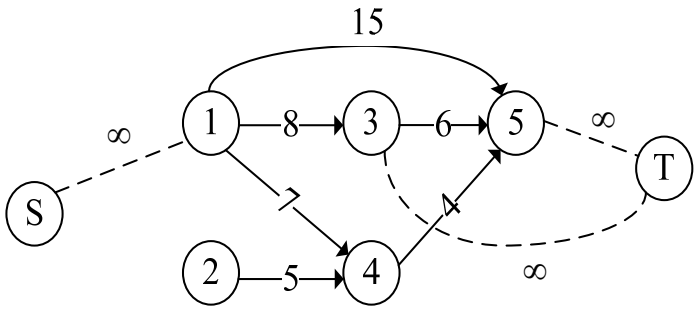


圖 19　Multiple Sources/sink Network 之範例圖

EX：

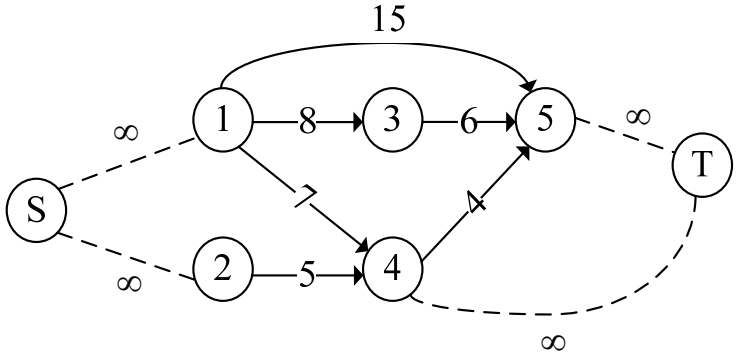
圖 19 為 Multiple Sources/sink Network 之範例圖，圖 19 中與節點 3 在同一路徑上的節點有節點 1 和節點 5；節點 1 在節點 3 之前，所以將它連到虛擬的 super source，而節點 5 在節點 3 之後，所以將節點 5 和節點 3 連到虛擬的 super sink；而與節點 3 不在同一條路徑上的節點 4，將它放入串列中，並利用 Relabel-to-front 演算法計算得知節點 4 push 4 單位的流量至 sink，再加上 source 直接流至 sink 的流量，得到 Mcut(3)=15+8+4=27；這也就是節點 3 通過的最大流量(mini cut)，如圖 20。

用同樣地的方法算出節點 4 的最大流量，得到 Mcut(4)=7+15+5+6=33，如圖 21；以此類推 Mcut(5)=15+6+4=25；最後 Mcut(0)、Mcut(1)都是沒有 in-edge，因此 Mcut(0)=0 及 Mcut(1)=0。



Mcut(3)=15+8+4=27

圖 20 應用於圖 19 範例之 Mcut(3)



Mcut(4)=7+15+5+6=33

圖 21 應用於圖 19 範例之 Mcut(4)

4.3 切割搜尋圖(The partition search graph)

切割搜尋圖(The partition search graph)有以下的特性：

1. 以 $\tilde{G}(\tilde{V},\tilde{E},\tilde{w})$ 表示之，$\tilde{V}$ 代表著所有節點的集合，$\tilde{E}$ 代表著邊的集合，$\tilde{w}$ 則為節點的權重。
2. $F_{in}(v)=\sum_{(u,v\in E)} w_{u,v}$ 定義成所有進入(in-coming)頂點 u 的總權重；$F_{out}(v)=\sum_{(u,v\in E)} w_{u,v}$ 則定義成由頂點 u 出去(out-coming)的總權重。
3. pred(V^l)表示在 (V^l, V^u)節點返回指標(backtrack pointer)到它的先行者節點。
4. M(U)為所有在 U 頂點集合中的的最小來源集合。

$$Scut(v^l \cup \{x\}, v^u \setminus \{x\}) = Scut(v^l, v^u) - F_{in}(x) + F_{out}(x) \qquad \text{公式七}$$

切割搜尋圖(The partition search graph)中，以一層接著一層(layer-by-layer)搜尋的方式擴張，切割上群組和下群組尋找一條最佳的路線。

首先在切割搜尋圖中，在 layer-0 對應成(0，0)；反之，layer-n 對應成($Ccut(V^l)$,0)，括號內的第一個數字在此代表此層至 layer-0 的累積成本($Ccut(V^l)$)，第二個數字則為 V^l 的分割權重($Scut(V^l, V^u)$)，而分割權重的算法如公式七。在考慮每一層之前，計算此層節點至 layer-0 的累積成本，除了在每層節點保留累積成本之外，我們也維護一個返回指標 (backtrack pointer)，此返回指標可以回到每個節點的最佳先行者(predecessor)的節點；當直至 layer-n 的($Ccut(V^l)$, 0)為止，此時|V|層群(V^l =V 且 V^u={})形成，最佳的頂點順序可以經由這個|V|層的返回指標(backtrack pointer) 沿著最佳路徑到它的先行者節點，找到 layer-0 的來源節點。

EX：將圖 19 的 DAG 圖，轉成切割搜尋圖且利用公式七，計算分割權重($Scut(V^l, V^u)$)，如圖 22。

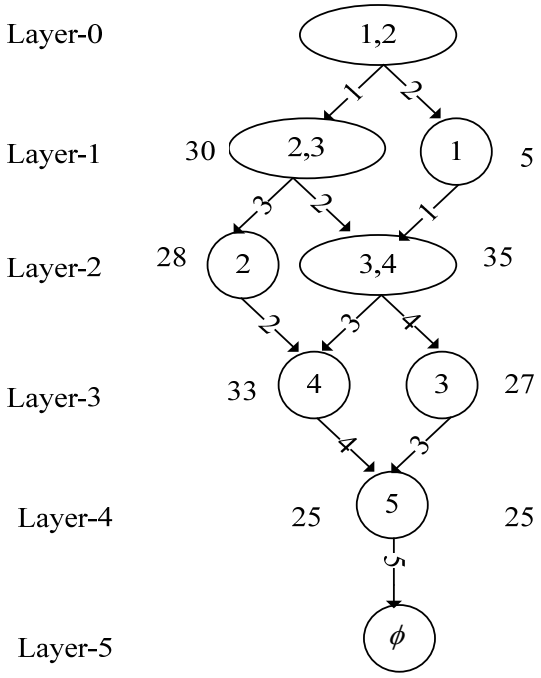


圖 22　切割搜尋圖和 $Scut(V^l, V^u)$

步驟一、首先將所有頂點先放至 V^u 中，而 $M(V^u)$置於 Layer-0 內。

步驟二、從 $M(V^u)$中選出其一頂點放至 V^l 中，利用公式七，算出每一個節點的從上群組中切割後的值。例舉 layer-0 來說明，從 $M(V^u)$ 中 $Scut(V^l, V^u)$ =0，在選擇點 1 後，$F_{in}(x)$ =0, $F_{out}(x)$=30，因此下群組的節點(2,3)的權重值為 30；依此類推，算出各節點的權重植。

步驟三、將每一層所得到的權重植，從上層累積下來，得到累積成本值，如下圖 23。

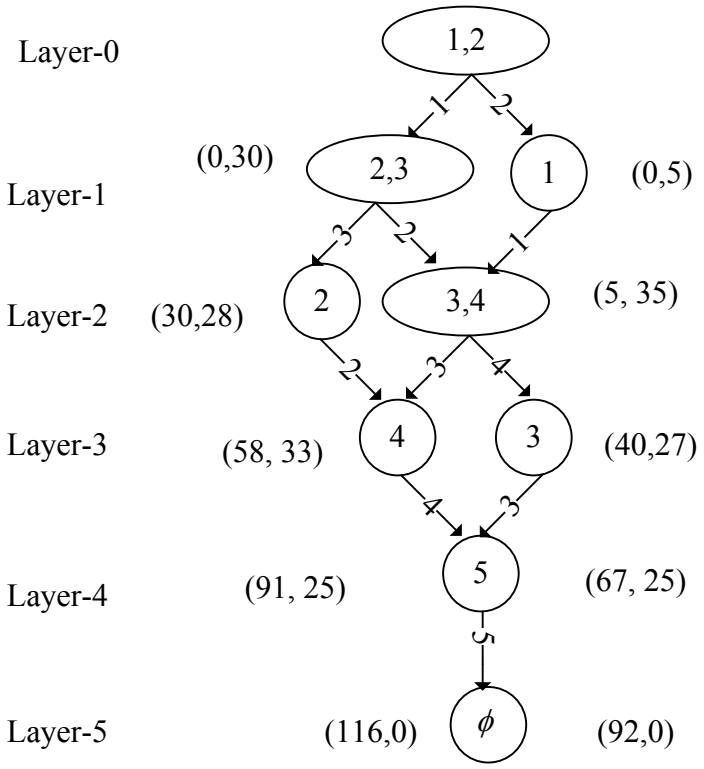


圖 23 $Ccut(V^l)$ and $Scut(V^l, V^u)$

4.4 Minimum Cut for Vertex Pair

在 Multiple Sources/sink Network 中，藉著上面所描述的方法，本論文提出一個可以提升 DAG 中的 Lower bound 方法，降低 A*搜尋法中開啟列表的節點數量，以尋找一條最佳路徑，得到一組最佳廣播序列。在此演算法中，我們只去考慮 joint vertex pair 的情況；而其它假設如同上述的演算法一樣。

首先在 DAG 圖中，我們先找出一對無相依性且有相同先行節點及相同後繼節點之頂點，稱之為 joint vertex pair，並將 joint vertex pair 視為一個集合節點，在這個集合節點中，針對這兩個節點做排序(哪個節點置於前面或置於後面)來考慮，比較算出的 Lower bound 是否比原來只有考慮單一個節點的 Lower bound 來的大，若把兩個節點視

為一個集合節點的 Lower bound 較大時，將增加的流量加到它們共同的先行節點的流量中，詳細步驟如下：

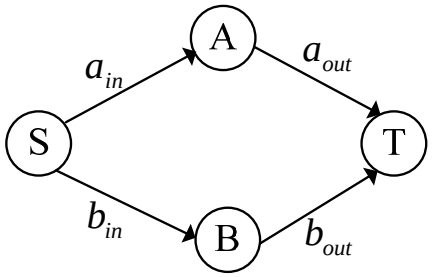


圖 24　Minimum Cut for Vertex Pair 之範例圖

步驟一：首先，分別算出單一的 Mcut(A) 以及 Mcut(B) 得到 Mcut(A)=$a_{in}+\min[b_{out},b_{in}]$ 以及 Mcut(B)=$b_{in}+\min[a_{out},a_{in}]$

步驟二：

i. 圖 24 中，節點 A 和節點 B 即為 joint vertex pair，因此將節點 A、B 視為一個集合節點，如果在集合節點中，A 排在 B 之前, 要計算節點 A 之 Mcut(A)，將節點 A 視為 sink 與 sink 連結，由於節點 B 在節點 A 之後，所以節點 B 也視為 sink 與 sink 連結，得到 Mcut(A)= $(a_{in}+b_{in})$。

ii. 若是計算節點 B 之 Mcut(B)時，將節點 B 視為 sink，但由於節點 A 在節點 B 之前，所以節點 A 則視為 source 與 source 連結，得到 Mcut(B)= $b_{in}+a_{out}$。

iii. 將 a、b 兩步驟求得值加總=>$(a_{in}+b_{in})+(b_{in}+a_{out})$。

步驟三：

i. 反之，如果把 B 排在 A 之前時，要計算節點 B 之 Mcut(B) 時，將節點 B 視為 sink 與 sink 連結，由於節點 A 在節點 B 之後，所以將節點 A 視為 sink 與 sink 連結，可以得到 Mcut(B)= $(a_{in}+b_{in})$。

ii. 若計算節點 A 之 Mcut(A)時，將節點 A 視為 sink，但由於節點 B 在節點 A 之前，所以將節點 B 視為 source 與 source 連結，得到 Mcut(A)= $a_{in} + b_{out}$ 。

iii. 將 a、b 兩步驟求得值加總=> $(a_{in} + b_{in}) + (a_{in} + b_{out})$.

步驟四：最後取步驟二、步驟三所產生的最小值，減去步驟一所算出的單一節點 Mcut(A)以及 Mcut(B)，得到的值就為增加的部份，如公式八。增加的值加到兩節點相同的前行節點中，可以得到 $Mcut^{(2)}(i)$ =Mcut(i)+Increase， $Mcut^{(2)}(i)$ 代表 Mcut(i)的前行節點。

$$\text{Increase}= (a_{in} + b_{in}) + \min\{\ [(a_{in} + b_{in}) + (b_{in} + a_{out})\ ], [(a_{in} + b_{in}) + (a_{in} + b_{out})\ ]\} - ([a_{in} + \min(b_{out}, b_{in})] + [b_{in} + \min(a_{out} + a_{in}\ ])$$

公式八

上述的步驟為一個簡單求得 Minimum Cut for Vertex Pair 的方式；此方法可延伸，並適用於任何有向無循環圖中，以增加其有向無循環圖中的 Lower bound。

EX：

沿用圖 19 說明更複雜情況下的 Minimum Cut for Vertex Pair。圖 19 中節點 3 和節點 4 為一對無相依性且有相同先行節點及相同後繼節點之頂點的，所以我們把節點 3 和節點 4 當成一個集合節點來看，此時會有兩種情況發生：

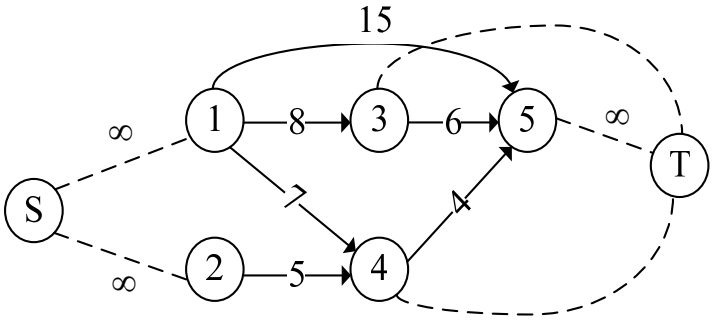
情況一(節點 3 放在節點 4 的前面)：

i. 如果先把節點 3 放在節點 4 的前面，要求得 Mcut(3)，如圖 25(a)；此時節點 3 和節點 4 會與虛擬的 super sink 連結，

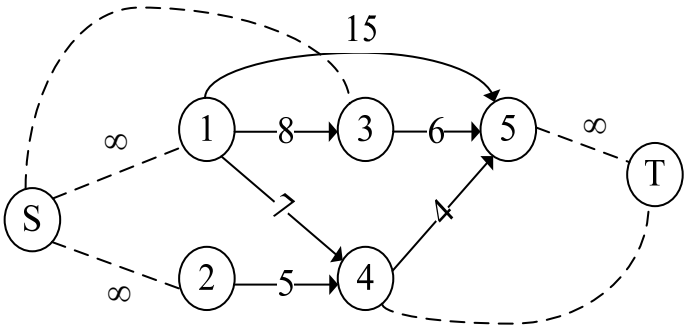
得到節點 3 的 Mcut(3)=8+15+7+5=35。

ii. 若計算 Mcut(4)值，如圖 25(b)，節點 4 與虛擬的 super sink 連結，但由於節點 3 是被在節點 4 之前的，所以會與虛擬的 super source 連結, 得到 Mcut(4)=15+6+7+5=33。

iii. 將 a、b 兩步驟求得值加總=>Mcut(3)+ Mcut(4)=68。



(a)　節點 3 放在節點 4 之前之 Mcut(3)



(b)　節點 3 放在節點 4 之前之 Mcut(4)

圖 25　Minimum Cut for Vertex Pair 情況一應用於圖 19 之範例

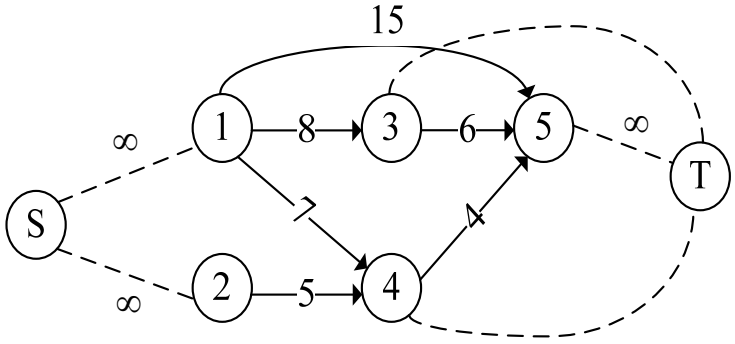
情況二(節點 4 放在節點 3 的前面)：

i. 接下來如果把節點 4 放在節點 3 的前面，欲計算 Mcut(4)值，如圖 25(a)；此時節點 3 和節點 4 會與虛擬的 super sink 連結，得到節點 4 的 Mcut(4)=8+15+7+5=35。

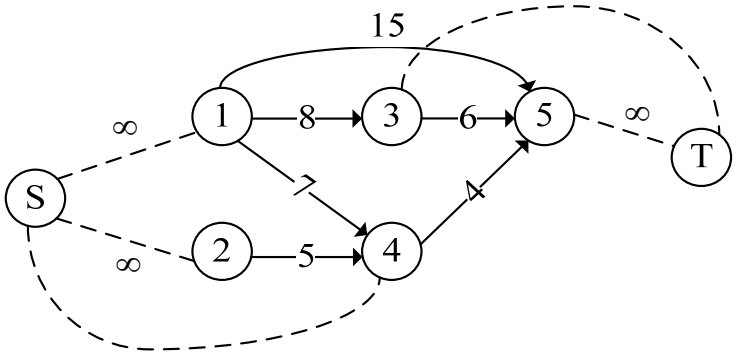
ii. 若欲計算 Mcut(3)值，如圖 26；節點 3 會與虛擬的 super sink 連結，但由於節點 4 是被在節點 3 之前的，所以會與虛擬

的 super source 連結, 得到 Mcut(3)=8+15+4=27。

iii. 將 a、b 兩步驟求得值加總=> Mcut(3)+ Mcut(4)=62。



(a)節點 4 放在節點 3 之前之 Mcut(4)



(b)節點 4 放在節點 3 之前之 Mcut(3)

圖 26 Minimum Cut for Vertex Pair 情況二應用於圖 19 之範例

最後取此兩種情況所產生的最小值，減掉原本是考慮單一情況的 Mcut(3)及Mcut(4)值。利用我們所提出的演算法所算出的最大流量至少增加了 62-60=2。從上面的實例看出，原本只考慮單一節點的 Mcut(3)+Mcut(4)=60。因此，本論文提出的方法，至少大概可以提升 3%(2/60)的Lower bound。

其實，也可將此想法延深至兩個獨立以上的節點，我們同樣地

考慮具有無相依性且有相同先行節點及相同後繼節點之三個頂點或四個頂點，將之視為一個集合頂點，並透過排序集合頂點內的頂點，來增加DAG中的Lower bound，而透過第5章針對joint vertex pair的實驗模擬顯示，相信此想法也能增加更多的lower bound於DAG中。

4.5 最大流量最小切割定理產生的下限解應用於A*圖層搜尋法：

在第二章文獻探討中，我們已詳細介紹，在此大略說明此搜尋法。在眾多找尋最小路徑問題中，A*搜尋法是最好的搜尋演算法之一，它使用一個啟發式的評估來預測；因此，可以快速地將許多明顯為壞的路徑排除考慮，節省記憶體的儲存空間，進而計算出一條滿意的路徑到目的地。A*搜尋法中，評估函數 $F(x)=G(x)+H(x)$，$G(x)$代表從來源節點到頂點x的成本。$H(x)$代表從頂點x到目標節點的估計成本。為了使A*搜尋法變得合理化，因此$H(x)$的估計必需謹慎決定。

在本論文，我們刪除 Mcut 值較小的節點，以降低 A*搜尋法中，Open list 內的數量；如此一來，分割搜尋圖中，每層的節點不需同時建立，藉此可快速找尋一條最佳路徑至目的地。而在此則利用累積成本來代入$G(x)$，$[\sum_{u\in V^u} Mcut(u)]-Mcut(x)=H(x)$ 來計算 A*搜尋法的評估函數。

以下為應用於A*圖層搜尋法之步驟：

步驟一、沿用圖19的DAG圖，轉成切割搜尋圖，承上節的圖23，算出每層的累積成本，此成本為$G(x)$。

步驟二、計算任一頂點的Mcut(x)，可得到Mcut(1)=0, Mcut(2)=0,

Mcut(3)=27, Mcut(4)=33, Mcut(5)=25，加總後得到 $\sum_{x\in V^u} Mcut(x) = 85$。

步驟三、算出每一個layer的 $[\sum_{u\in V^u} Mcut(u)] - Mcut(x)$ 值，此值為 $H(x)$；用來估計至目的地的估計成本。如圖27

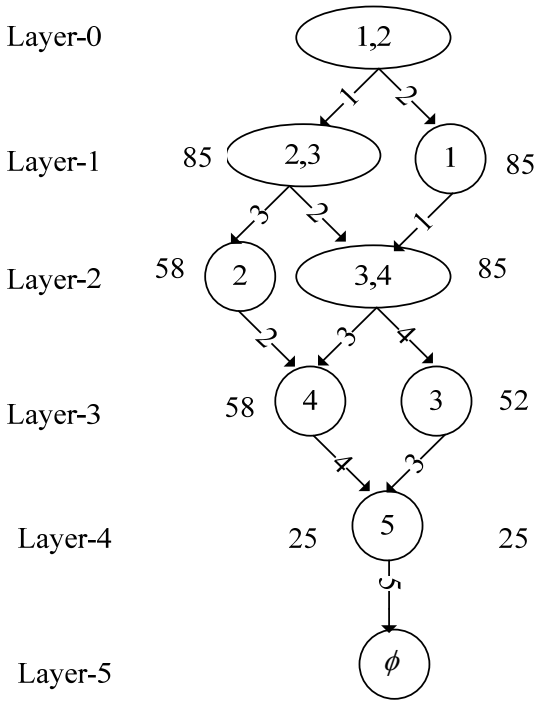


圖 27 切割搜尋圖和 $H(x)$

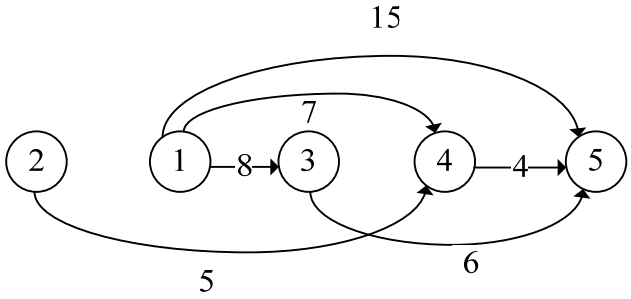


圖 28 利用 $W(f)$=98 為上限值

步驟四：

i. 假設圖28為一拓撲排序之邊的總延遲成本，將此值視為上限值。

ii. 用 $F(x)=G(x)+H(x)$，算出每條路徑的成本。節點旁邊的數字為 $F(x)$ 值。若大於上限值的路徑，則不考慮。

如圖29中虛線表示。

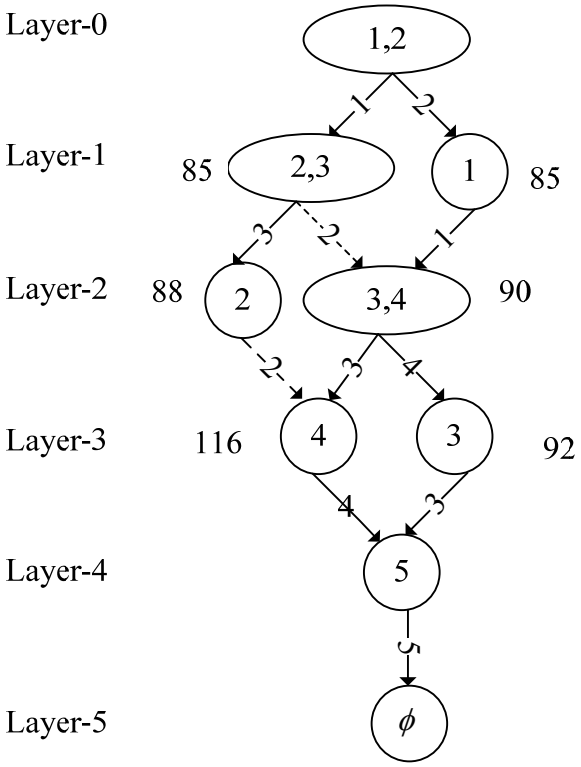


圖 29　切割搜尋圖和 $F(x)$

步驟五：在 Layer-3，Mcut(3)<Mcut(4)，因此將節點 3 從 Open list 中刪除。如圖 29。

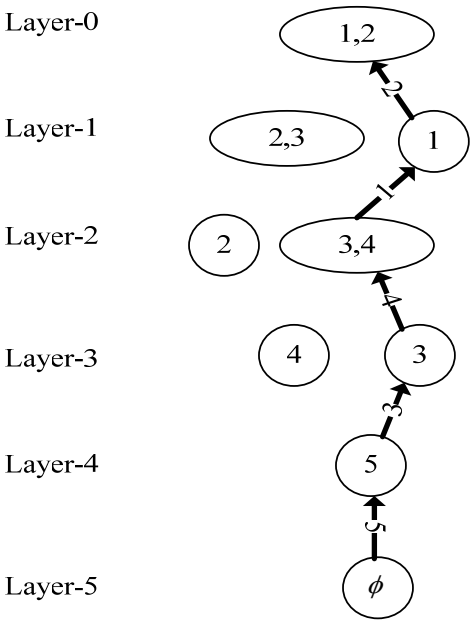
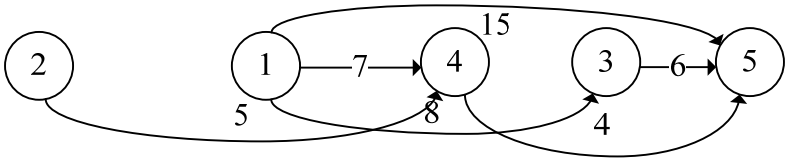


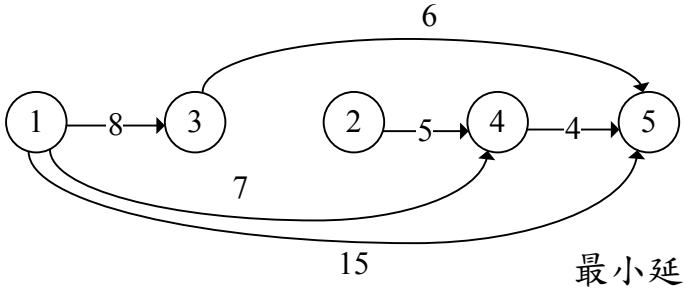
圖 30　利用於 A*圖層搜尋法之最佳路徑

步驟六：此時|V|層群(V^{l} =V 且 V^{u}={})形成，最佳的頂點順序可以經由這個|V|層的返回指標(backtrack pointer) 找到 Layer-0，因此最佳路徑用粗線表示為 2->1->4->3->5。如圖 30。

步驟七：將上述步驟所得出的最佳路徑使用拓撲排序，如圖31(a)，得到$W(f)$為5*2+8*2+4*2+7+15*3+6=92，與圖31(b)、31(c)、31(d)比較，圖31(a)的總延遲成本為最小，因此，圖31(a)此組排序為最佳排序。

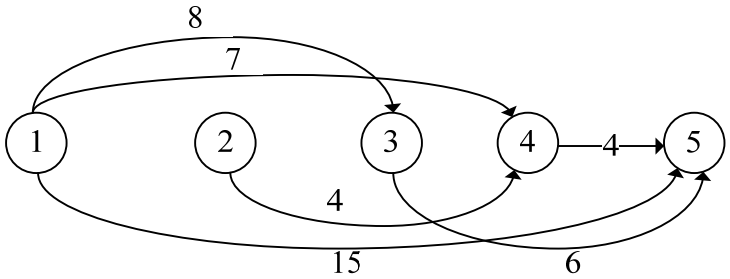


(a)　最佳的拓撲排序



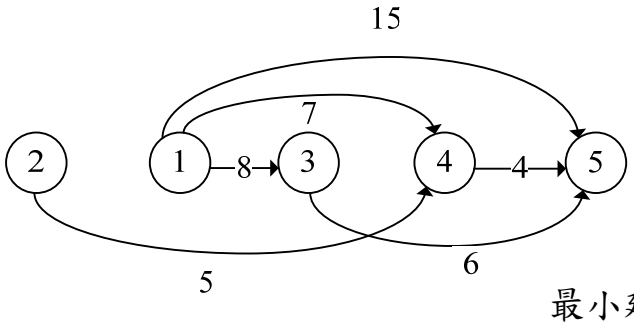
最小延遲成本為8+7*3+15*4+6*3+5+4=116

(b)



(c)

最小延遲成本為8*2+7*3+4*2+4+15*4+6*2=121



最小延遲成本為5*3+6*2+8+7*2+15*3+4=98

(d)

圖 31　拓撲排序

第五章 模擬實驗

為了評估本論文提出利用 joint vertex pair 解決廣播排程問題的效能，我們執行模擬實驗來進行分析。5.1 節為模擬環境的介紹，5.2 節為實驗資料檔的產生與介紹，5.3 節為實驗結果分析。

5.1 模擬環境

在進行模擬實驗中，我們使用作業系統為 Microsoft Windows XP，利用 JAVA 程式語言來開發模擬程式。在模擬環境中，我們假設用戶端查詢的資料項大小固定、長度相等；處理用戶端查詢的能力為非即時的，它會在一段時間內收集用戶端所要求之資料項之後，將資料項根據本論文所提出的演算法進行排程，產生出一組較佳的廣播序列，透過廣播頻道將資料項傳送出去滿足用戶端需求，並降低用戶端的平均查詢讀取時間。

5.2 實驗資料檔的產生與介紹

經由造檔程式產生出進行模擬實驗所需的資料檔案，模擬用戶端對於伺服端提出資料項查詢的情形。在本文的實驗資料檔中，每一實驗資料檔中，假設存在 50 個資料項，兩資料項間的相依程度(即權重)；此相依程度利用公式九計算。實驗數據則執行 10 個實驗資料檔求其平均。

$$w^{(i)} = \frac{i^{-s}}{\sum_{j=1}^{m} j^{-s}} \qquad 1 \le i \le m$$

公式九

m：用戶端的查詢數(100~300)

S：Skew 偏斜度(S=0, S=1.0, S=2.0)

每一個實驗資料檔中，任一查詢的存取頻率使用不同的 Skew 參數在 Zipf 分配下執行實驗。Zipf 分配(Zip f distribution)用來假設用戶端讀取資料項的機率以及調整資料項被存取機率的偏斜程度，當θ等於 0 的時候為均勻分配(uniform distribution)，代表每個資料項被存取的機率相似;θ 越高時，表示某一部分資料項被存取的機率特別高，用戶端存取的頻率越不平均。

將產生出的資料檔案利用最大流量最小切割定理產生的下限解應用於 A*圖層搜尋法方式的拓樸排序演算法，獲得的廣播排程結果相比較，試著找尋到較低的平均資料相依長度，以降低用戶端平均查詢讀取時間。

5.3 實驗結果分析

本論文將透過實驗來評估在問題解決上所使用的各個演算法效能的差異。以用戶端收取資料項所花費的平均查詢讀取時間為標準，分別以不同的用戶端查詢數、不同的查詢偏斜程度以及廣播資料項總數來進行模擬實驗。

首先以偏斜程度為 0 的均勻分配下，做不同查詢數量分別為 100、150、200、250、300 之下做演算法效能的比較。

表一、表二、表三中顯示出不同查詢數量下用戶端花費的平均查詢讀取時間之變化，結果顯示出平均查詢讀取時間會隨著查詢數增加相對的提高，圖 32、33、34 分別為偏斜度為 0、1.0、2.0 的情況，從圖中清楚地看出本論文所提出的方法，可成功地提升原本的 Lower bound ，使得結果更為接近最佳值。

表 1 偏斜度 S=0 以及不同查詢數量下，演算法的平均查詢讀取時間

查詢數量 / 演算法	**100**	**150**	**200**	**250**	**300**
opt	8.77	11.53	12.65	13.82	14.41
Mcut	5.56	8.95	10.18	9.742	8.40
Lower bound	2.00	2.58	3.21	3.90	4.55

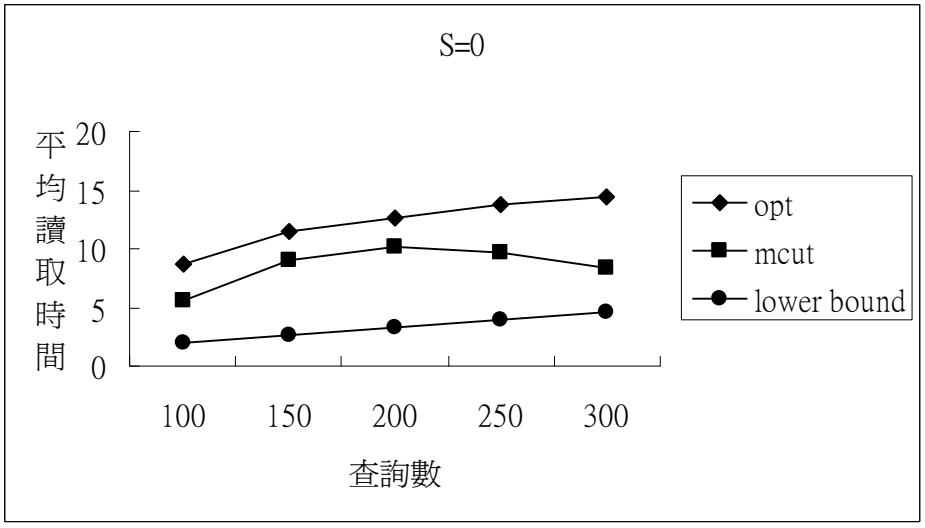


圖 32 偏斜度 S=0 及依照不同 query 下用戶端的平均查詢讀取時間

表 2 偏斜度 S=1.0 以及不同查詢數量下，演算法的平均查詢讀取時間

查詢數量 / 演算法	**100**	**150**	**200**	**250**	**300**
opt	6.15	8.74	9.73	10.34	10.73
Mcut	4.47	7.00	7.97	7.24	6.08
Lower bound	1.75	1.97	2.21	2.44	2.75

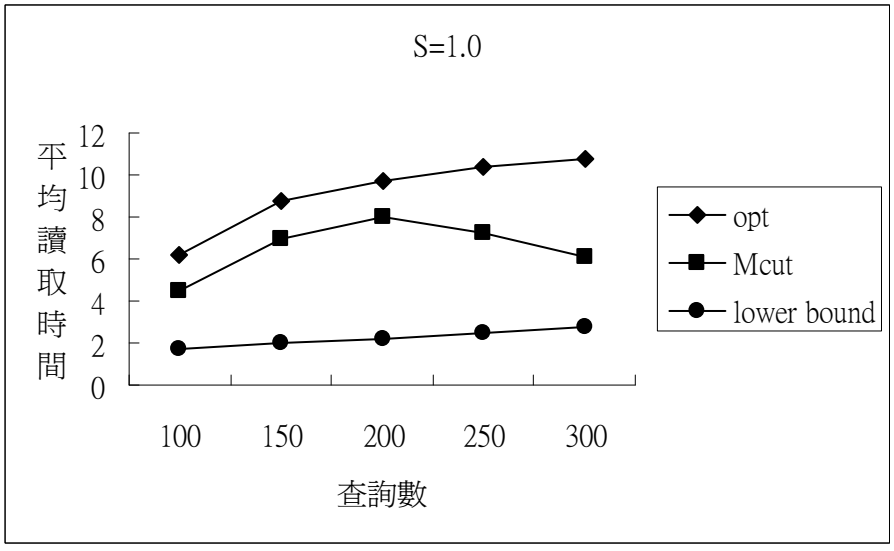


圖 33 偏斜度 S=1.0 及依照不同 query 下用戶端的平均查詢讀取時間

表 3 偏斜度 S=2.0 以及不同查詢數量下，演算法的平均查詢讀取時間

查詢數量 / 演算法	100	150	200	250	300
opt	4.36	7.62	8.53	8.95	9.25
Mcut	3.97	6.68	7.43	6.49	5.15
Lower bound	2.69	1.94	2.12	2.16	2.35

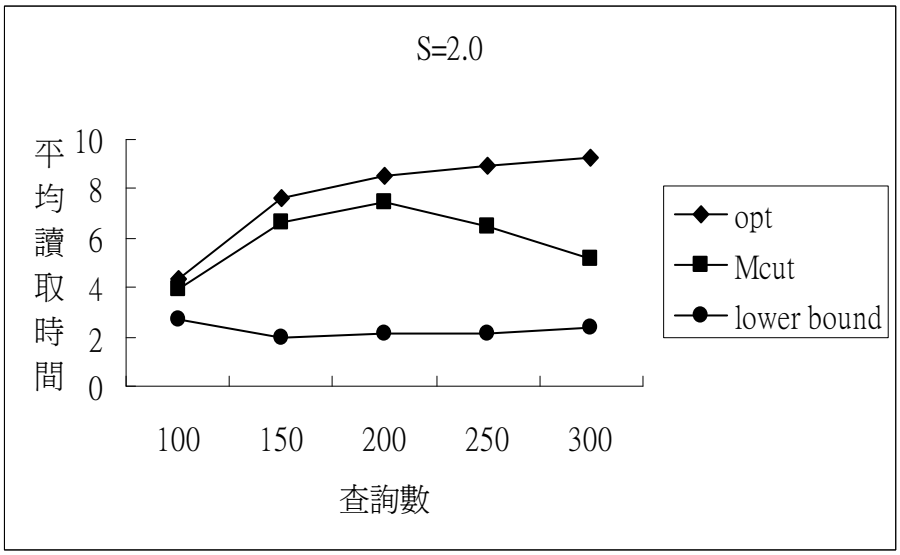


圖 34 偏斜度 S=2.0 及依照不同 query 下用戶端的平均查詢讀取時間

另外表四及圖 35 顯示出不同查詢數量下用戶端所增加之百分比變化，結果顯示提升的百分比會隨著查詢數增加相對的降低，加入 increase lower bound 的方法後，在偏斜度為 0、1.0、2.0 的情況，大致上可提升將近快 3% ；偏斜度為 1.0 且查詢數量為 100 時甚至提升接近 9%。

表 4 在不同偏斜度以及不同查詢數量下，增加 lower bound 之百分比

查詢數量 / 偏斜度	**100**	**150**	**200**	**250**	**300**
S=0	7.2%	4.5%	3.8%	2.1%	1.2%
S=1.0	8.6%	5.5%	4.8%	2.8%	1.2%
S=2.0	2.5%	2.8%	2.6%	1.1%	0.3%

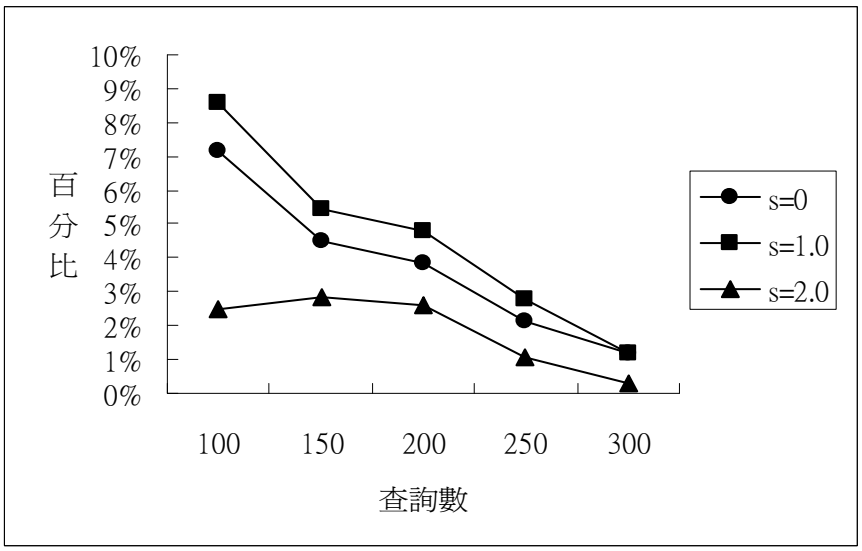


圖 35 不同偏斜度以及不同查詢數量下，增加 lower bound 之百分比

第六章 結論

本論文以無線廣播中廣播資料項之間存在的相依性為主要考量，使用有向無循環圖形(DAG)表示其順序限制，而每一資料項對應於圖形中每一頂點；我們將線性排列的問題，用圖形搜尋來考慮，欲求線性排列的最佳解，本論文利用 Relabel-to-front 演算法計算 DAG 中每個節點的 minimum cut，並且提出 joint vertex pair 方法，更進一步提升 DAG 圖中的 Lower bound；最後，應用於 A*圖層搜尋法中，以降低搜尋路徑時，記憶體的暫存空間，提供更快速更佳的路徑解。

經由實驗結果顯示，利用 A*圖層搜尋法，可得到一組最佳的拓撲排序，確實可以縮短資料項之間的相依性長度、降低頂點之間平均路徑長度，減少用戶端在收取所需資料項時花費的平均查詢讀取時間。

未來，可將此想法延深至兩個獨立以上的節點，同樣地考慮具有無相依性且有相同先行節點及相同後繼節點之三個頂點或四個頂點，將之視為一個集合頂點，並透過排序集合頂點內的頂點，來增加 DAG 中的 Lower bound，透過第 5 章針對 joint vertex pair 的實驗模擬顯示，相信此想法也能增加更多的 Lower bound 於 DAG 中。

參考文獻

[1] D. Aksoy and M. S. F. Leung, “Pull vs Push: A Quantitative Comparison for Data Broadcast,” Global Telecommunications Conference, 2004.

[2] G. Alteka and D.A. Joseph, ”CS270 Combinatorial Algorithms & Data Structures,” 2005.

[3] A. Bar-Noy, J. Naor. And B. Schieber, “Pushing dependent data in clients-providers-servers systems,” Wireless Networks, 2003.

[4] G. Chartrand and O. R. Oellerman, “Applied and Algorithmic Graph Theory,” McGraw-Hill, 1993.

[5] C. Chehadeh, A.R. Hurson and L.L. Miller, “Energy-efficient indexing on a broadcast channel in a mobile database access system,” IEEE, Proceedings of the International Conference on Coding and Computing Information Technology, 2000.

[6] D. Chung and M.H. Kim, “An index replication scheme for wireless data broadcasting,” ACM, Journal of Systems and Software, 2000.

[7] S. Chen, K.L. Wu and P.S. Yu, “Optimizing Index Allocation for Sequential Data Broadcasting in Wireless Mobile Computing,” Transaction on Knowledge and Data Engineering, 2003.

[8] Y. Chung and M.H. Kim, ”On Scheduling Wireless Broadcast Data,” KAIST Department of Computer Science, 1998.

[9] C. Ding, X.He, H.Zha,M.Gu, H.Simon, ”A Min-max cut algorithm for graph partitioning and data clustering,” IEEE 1st Conference on Data Mining,2001.

[10] O.E. Demir and D. Aksoy, “Energy-Efficient Broadcast-based Event Update Dissemination,” Pertormance, Computing and Communications, 2004 IEEE International Conference, 2004.

[11] Q. Fang, d S.V. Vrbsky, Y. Dang and W. Ni, “A Pull-Based

Broadcast Algorithm that Considers Timing Constraints," Proceedings of the International Conference on Parallel Processing Workshops, 2004.

[12] J. Huang and Y. Lee, "Efficient Index Caching Schemes for Data Broadcasting in Mobile Computing Environments," Proceedings of the 14th International Workshop on Database and Expert Systems Applications, 2003.

[13] M.A. Iverson and Fusun Ozguner and Gregory J.Follen, "Parallelizing Existing Applications in a Distributed Heterogeneous Environment," NASA Lewis Research Center Grant.

[14] T. Imielinski, S. Viswanathan and B.R. Badrinath, "Energy efficient indexing on air," ACM, Proceedings of Special Interest Group on Management of Data Conference, 1994.

[15] K.F. Jea and M.H. Chen, "A Data Broadcast Scheme Based on Prediction for The Wireless Environment," Proceedings of the Ninth international Conference on Parallel and distributed Systems, 2002.

[16] A.B. Kahn, "Topological Sorting of Large Networks," Communications of the ACM, 1962.

[17] G. Lee and M.S. Yeh and S.C. Lo and A. L. Chen, "A Strategy for Efficient Access of Multiple Data Items in Mobile Environments," Proceedings of the Third International Conference on Mobile Data Management, 2002.

[18] K.F. Lin and C.M. Liu, "Schedules with Minimized Access Latency for Disseminating Dependent Information on Multiple Channels," Proceedings of the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing, 2006.

[19] K.Y. Lu and G.M. Wu, "Relaxation-Based Data Placement for Wireless Broadcast," IEEE 6th CAS Symp on Emerging

Technologies: Mobile and Wireless Comm, 2004.

[20] S.C. Lo and A.L.P. Chen, "Optimal index and data allocation in multiple broadcast channels," IEEE, Proceedings of 16th International Conference on Data Engineering, 2000.

[21] W. Ni, Q. Fang and S.V. Vrbsky, "A Lazy Data Request Approach for On-demand Data Broadcasting," Proceedings of the 23rd International Conference on Distributed Systems Workshops , 2003

[22] O.B.V. Ramanaiah and H. Mohanty, "NICD: A Novel Indexless Wireless On-Demand Data Broadcast Algorithm," Proceedings of the Internation Conference on Information Technology: Coding and Computing , 2004.

[23] N. Saxena, K. Basu and S.K. Das, "Design and Performance Analysis of a Dynamic Hybrid Scheduling Algorithm for Heterogeneous Asymmetric Environments," Proceedings of the 18th International Parallel and distributed Processing Symposium , 2004.

[24] R. Sakellariou and H. Zhao, "A Hybrid Heuristic for DAG Scheduling on Heterogeneous Systems," Proceedings of the 18th International Parallel and Distributed Processing Symposium, 2004.

[25] M. Tarique, K.E. Tepe, and Mohammad Naserian, "Energy Saving Dynamic Source Routing for Ad HocWireless Networks, "

[26] X. Wu, V. C. S. Lee, "Preemptive Maximum Stretch Optimization Scheduling for Wireless On-Demand Data Broadcast," Proceeding of the International Database Engineering and Applications Symposium, 2004.

[27] J. Xu, B. Zheng, W. C. Lee and D. L. Lee, "Energy Efficient Index for Querying Location-Dependent Data in Mobile Broadcast Environments," Proceedings of the International Conference on Data Engineering , 2003.

[28] http://en.wikipedia.org/wiki/Relabel-to-front_algorithm,

"Relabel-to-front algorithm,"

[29] http://en.wikipedia.org/wiki/A-star_search_algorithm,

"A* search algorithm,"

[30] http://en.wikipedia.org/wiki/Dfs,

"DFS algorithm"