旋轉圖之互斥問題演算法

林宏仁
國立台北商業技術學院商學研究所
tomlin@mail.ntcb.edu.tw

林佳蓉
國立台北商業技術學院商學研究所
cindy006@gmail.com

摘要

在分散式系統架構中，互斥是一個重要且基礎的問題，在過去已有許多在不同網路拓樸下提出解決互斥問題的研究被發表。本文將提出在旋轉圖中解決互斥問題的演算法。旋轉圖是一種廣受注意的網路拓撲架構，其具有許多優良的拓撲特性，諸如較小的直徑、遞迴建構及唯一的最短繞徑等。我們利用其拓撲特性，在節點數為 m 的旋轉圖上，發展出二種互斥問題演算法。第一個演算法將一個旋轉圖分成許多小的獨立單位，配合前置計算，將該單位對應到矩陣資料結構中，使得法團大小接近 $2\sqrt{m/6}$ 。另一個演算法則利用旋轉圖具有唯一最短路徑衍生樹的特性，配合衍生樹中節點編號排列的規則，不需前置計算，只需知道發起詢問的節點編號，即可進行該互斥演算法。

關鍵詞：分散式系統、互斥、旋轉圖、法團、最短路徑衍生樹

Abstract

Mutual exclusion (ME) is a basic and also an important issue in distributed systems. In the past, there were many published papers that provide algorithms for ME problems in different topologies of interconnection networks. This paper provides algorithms to solve ME problem in rotator graphs. A rotator graph, which has many good properties such as low diameter, recursive construction and unique shortest path routing, is a well-known topology and is widely published in recent researches. We utilize its topology properties to develop two different mutual exclusion algorithms. In a rotator graph with node size of m, the first algorithm divides the whole graph into many distinct small units and maps them to the nodes of a mesh structure by a pre-compute process, which accomplishes the result of the quorum size to be close to $2\sqrt{m/6}$. The second algorithm, which utilizes the property of unique shorts path spanning tree of a rotator graph and the permutation property of nodes, accomplishes mutual exclusion algorithm with the information of source node permutation, without pre-compute process.

Keywords: distributed system, mutual exclusion, rotator graph, quorum, shortest path spanning tree

1. 簡介

隨著電腦的普及網路的發展，分散式計算的架構逐漸成為企業資訊系統的重要發展方向。分散式系統是由一些分散在各地的處理單元所組成，藉由網路聯合彼此間的儲存資源、運算能力及容錯備援，以因應各種在實務運作上可能發生的複雜問題。在分散式系統中，經常以某些圖形做為其拓撲架構，並以這些拓樸的特性解決分散式系統的各項議題，如互斥(mutual exclusion)、同步(synchronization)、一致性(consistency)及資源分配(resource allocation)等。這些拓撲架構可能為處理器間的實體網路連接，或是建構在某實體連接之上

的抽象互連網路。本論文研究的旋轉圖(rotator graph)，是一種廣受注意並有許多相關研究的網路拓撲架構[3]。它具有許多優良的拓撲特性，諸如較小的直徑、遞迴建構及唯一的最短路徑衍生樹等。這些拓樸特性，使得在旋轉圖上進行繞徑(routing)、容錯路徑(fault tolerance paths)、群播(multicasting)及廣播(broadcasting)等問題能有效率解決[1, 4, 5, 6, 7, 8]。

在分散式系統中，由於採用多部電腦透過網路連結、互相溝通訊息及分攤工作，經常必須共享各種網路資源，也因此衍生出共享資源的互斥問題。所謂互斥問題(mutual exclusion)意指在分散式系統下，某些資源不允許同時被二部以上的電腦同時取用，即在同一個時間內只允許一部電腦使用該資源。許多分散式系統的操作包括資料複製及分散式共享記憶體等都需要解決共享資源的互斥問題，以保證備份資料的一致性，或是資源使用時的互斥性。互斥問題的解決方式通常可分為集中式及分散式處理。以集中式處理來說，當某一電腦需要使用獨佔資源，或者我們稱之為要進入臨界區間(critical section, CS)時，則固定向某一部主機發出請求使用訊息，並由該主機確保資源的獨佔使用原則，適時回覆該資源的使用許可。集中式在設計及管理上固然較為容易，但是卻缺乏容錯度，即當該主機發生錯誤時，整個分散式系統亦將停擺。有別於集中式處理，在分散式處理中，系統可能不存在一部主控主機，各電腦的權力相同；當一部電腦要求進入 CS 時，必須詢問其他電腦，藉以取得使用獨佔資源的權力。然而，為了保證執行效率，必須詢問的電腦個數及發送的詢問封包個數經常是在設計演算法時的重要考慮因素。本論文提出的演算法，以分散式處理為基礎建構。

過去的研究之中，旋轉圖已經被用來解決許多分散式系統的問題，但在互斥問題的研究上，尚未有相關的研究出現。因此，本論文提出在二種在旋轉圖中的互斥問題演算法。第一個演算法將一個具有 m 個節點旋轉圖分成許多較小的獨立單位，配合矩陣資料結構，使得法團大小(quorum size)接近 $2\sqrt{m/6}$ 。另一個演算法則利用旋轉圖具有唯一最短路徑衍生樹(shortest path spanning tree)的特性，配合衍生樹中節點編號排列的規則，使得進行詢問時封包可直接沿著最短路徑衍生樹的某一子樹的邊傳送，而不需前置計算即可散布詢問請求。

2. 圖形定義

一個規模為n的旋轉圖，在本文中以R_n表示，具有$n!$個節點，並且任意節點$S = s_1s_2\ldots s_n$有一個由符號 1, 2, 3, ... , n所組成的唯一排列。由某一節點連向另一節點的邊g_i，是由以下的生成函數所定義：

$$s_1s_2\ldots s_n \xrightarrow{g_i} s_2s_3\ldots s_is_1s_{i+1}\ldots s_n$$

由以上的生成函數可知，除了g_2邊可以雙向連絡兩個節點，其他在旋轉圖的邊皆具有方向性。例如，節點 123 是規模為 3 的旋轉圖其中一節點，其向外的連結分別是經由g_2連往節點 213 和經由g_3連往節點 231。因為旋轉圖是一個對稱圖，因此，在R_n中的每個節點的外分支度 (out-degree) 及內分支度 (in-degree) 均為 n-1。圖 1 為規模 3 的旋轉圖，圖 2 則為規模 4 的旋轉圖。很明顯的，一個R_4是由 4 個獨立的R_3組合而成。此遞迴建構 (recursive construction)的特性在任何規模的旋轉圖中均存在，即一個R_n是由n個獨立的R_{n-1}組合而成。

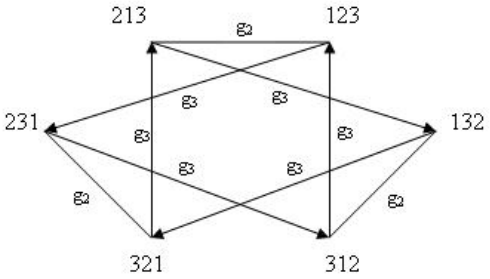

圖 1　旋轉圖R_3

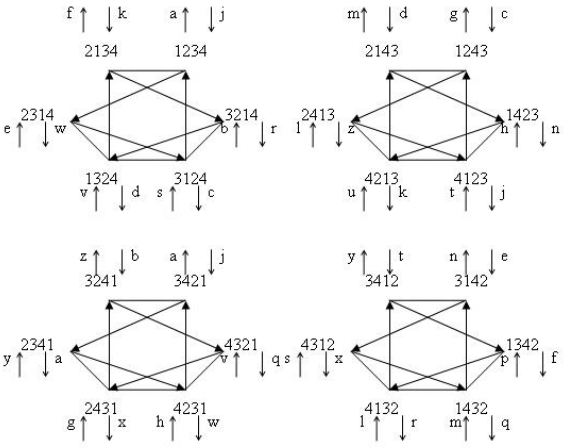

圖 2　旋轉圖R_4

在旋轉圖R_n中的各個節點均有一個節點編號，並且其編號為唯一的符號排列，因此我們稱具有 12…(*n*-1)*n*符號排列的節點為起始節點。我們以p(*i*, S)表示符號*i*在節點S的符號排列中的位置。例如，假設S=435126，則p(3, S) = 2 且p(5, S)=3。而R_n中的各個節點均具有一個有序數列(ordered sequence)，若其滿足p(*n*, S) > p((*n*-1), S) > … > p((*k*+1), S) > p(*k*, S) 且p(*k*, S) < p((*k*-1), S)，則我們稱其擁有長度為$n - k + 1$ 的有序數列。若$k = 1$，則其有序數列長度為 *n*，即為起始節點。

定理 2-1. 假設S為R_n中的一個節點，且非起始節點，若p(*n*, S) > p((*n*-1), S) > … > p((*k*+1), S) > p(*k*, S) 且p(*k*, S) < p((*k*-1), S)，則其有序數列為$n - k + 1$ 且從起始節點到S的最短路徑長度為$k - 1$。

証明：若節點 S 具有 p(*n*, S) > p((*n*-1), S) > … > p((*k*+1), S) > p(*k*, S) 且 p(*k*, S) < p((*k*-1), S)的性質，表示在 S 的節點編號中，符號 *k*，*k*+1，…及 *n* 依序由左至右排列(不一定均相連)，但符號 *k*-1 排列在 *k* 的右方(不一定相連)。若從起始節點 123…*n* 開始，每次由其邊的函式調整最左邊的符號至右方第 2 至 *n* 的位置，且因為是最短路徑走法，每個邊只需調整一次，因此共調整符號 1，2，…，及(*k*-1)即可到達節點 S，共調整 *k*-1 步。 □

例如，假設節點 S=215346，因 p(6, S) > p(5, S) 且 p(5, S) < p(4, S)，可知起始節點到節點 S 的最短路徑長度為 5 – 1 = 4，即 123456 → 234156 → 342156 → 421536 → 215346。

定理 2-2. 當樹根節點固定時，R_n具有唯一之最短徑衍生樹。

証明：R_n的最短路徑衍生樹是R_n的子圖，其包括R_n的所有節點，以及由一選定的節點(樹根)到各節點的最短路徑邊組成。因為在R_n中，某一節點到另一節點的最短路徑為唯一，因此該最短路徑衍生樹亦是唯一。 □

3. 互斥問題演算法

在本節中，我們將提出二種在旋轉圖架構下發展的互斥問題(mutual exclusion)演算法。第一種方法利用旋轉圖遞迴建構的特性，將一個R_n分成*n*!/3!個獨立的R_3，並將這些獨立的R_3分配到一個矩陣(mesh)的資料結構中，利用矩陣的一組行(column)與列(row)必與另一組行與列存在交集的特性，達到互斥的目的。另一種演算法利用旋轉圖最短路徑衍生樹(shortest path spanning tree)的特性，設計衍生樹的特定子樹與樹根的節點個數總合超過所有節點數的一半，達成單一互斥的目的。

3.1 以矩陣為基礎之互斥問題演算法

在非集中式的分散式系統中，任一節點請求使用獨佔資源時，必先發出請求並取得其他節點同意後才可使用該資源。因此，發出請求的節點個數是決定演算法效率的關鍵因素之一。Maekawa已證明在節點數為m時，法團(quorum)大小為$\sqrt{m}$是分群(coterie)的最佳解[2]。我們利用旋轉圖的遞迴建構的特性，將一個R_n分成$n!/6$ 個獨立的R_3，並將每一個獨立的R_3對應到一個矩陣結構(mesh)的一個節點中。由於一個矩陣的任一行(column)及列(row)的組合必定與另一行列組合存在交集，因此本節的演算法利用此項特性，在一個節點請求使用獨佔資源時，先計算該節點在對應矩陣架構中節點的座標，再依序請求所屬該行及列中所有節點的同意，才可使用獨佔資源。而本方法在R_n中法團大小接近$2\sqrt{n!/6}$。

定義 3-1. 一個旋轉圖中的子圖稱為一個獨立R_3，意指在該子圖中，各節點僅以邊g_2及g_3即可互相連通。

定理 3-1. 在一個旋轉圖R_n中，存在$n!/6$ 個節點不重覆的獨立R_3。

證明：旋轉圖具有遞迴建構的特性，即一個R_n是由n個獨立的R_{n-1}組合而成，例如R_4具有 4 個R_3。因此，每一個R_n都可以分解成$n*(n-1)*(n-2)*\ldots*4$ 個獨立的R_3。因此，R_n共可分解成$n!/6$ 個不重覆的獨立R_3。 □

例如，R_6共有 6!個節點，且可被分解為 6!/6 = 120 個R_3。

具有相同n-3 個符號排列結尾的節點屬於一個獨立R_3，意指前述的 6 個節點可以僅使用g_2及g_3兩種邊互相連通。例如，在R_5中，節點12345、13245、21345、23145、31245及32145屬於同一個獨立R_3。

定理 3-2. 一個獨立R_3中的任一節點，可經由路徑$g_3g_2g_3g_2$依序拜訪該獨立R_3的另外 3 個節點，並回到節點p。

證明：由*定義 3-1*，我們知道在R_3中各節點僅以邊g_2及g_3即可互相連通，且R_3中僅以三個符號排列。因此，任一節點經由路徑g_3g_2即可走到自己的反序節點，再重覆一次g_3g_2繞徑，便走回本身節點。 □

例如，節點 231 經由g_3g_2繞徑，可走至 132，為 231 之反序節點，再重覆走g_3g_2繞徑，即可回到自己本身節點，其路徑為：231 → 312 → 132 → 321 → 231。

在本節的互斥演算法中，我們將旋轉圖的每一個獨立R_3對應到一個矩陣架構的各個節點中，左上角R_3座標為(1, 1)。如同*定理 3-2* 所述，我們只要經由$g_3g_2g_3g_2$的繞徑，即可拜訪包括發起者在內的 4 個節點。因為一個獨立R_3共有 6 個節點，若有 4 個節點全部同意獨佔資源的使用，則該獨立R_3則不可能再有另外不同的 4 個點可以同意其他人的獨佔權，我們以此確保互斥的正確性。在以下的演算法中，我們提出關於計算對應矩陣大小及對應矩陣座標的方法。

演算法 3.1 計算一個R_n的對應矩陣大小。

Procedure compMesh(節點規模 n)

步驟：

1. 計算在R_n中R_3的數量$x = n!/6$.
2. 計算$\sqrt{x}$的值，若其為整數則對應矩陣大小為$\sqrt{x}$ *$\sqrt{x}$；若不為整數，則求得最接近的二個整數 m、l 相乘之乘積為 x，則對應矩陣大小為 $m * l$。

end procedure

演算法 3.2 計算一個獨立R_3對應於矩陣的座標。

Procedure compXY(獨立R_3的尾序)

步驟：

1. 利用以行為主(column major)的方式計算R3 位於矩陣中的順序，其中 k=n-3：

   $seq = \sum_{i=1}^{k}[(pi-1)\times\prod_{j=i}^{k-1}(n-j)]$ 。

2. 令座標=(x, y)，並且以 *floor*()以及 *ceil*()分別表示 *floor* 及 *ceiling* 函式：

   $x = seq - m$ * floor(seq / m)

   y = ceil(seq / m)

end procedure

接下來，我們設計二個演算法，分別為：getArgeement(*演算法 3.3*)以及request(*演算法 3.4*)。getAgreement是一個子程序，用以取得某一個R_3內任意 4 個節節點的同意。而request則是當有節點欲取得獨佔資源時，發出請求時所使用的程序。在request中，欲取得獨佔資源的節點，必須先取得自己所屬R_3的同意，再取得同一列以及同一行所有R_3群組的同意，方能進入臨界區間。而當某節點接收到request時，則利用最短路徑回覆同意與否之訊息(*演算法 3.5*)。演算法如下所示：

演算法 3.3 詢問一個獨立R_3的同意權

Procedure getAgreement(節點 p)

p 為發出詢問之節點。

初始值：*agree*=true;

步驟：

1. 詢問節點p設定一詢問封包，依詢問路徑$g_3g_2g_3g_2$依序傳送。
2. 當接受詢問節點已經同意其他節點使用獨佔資源，則設定 *agree*=false。
3. 當封包回到p後，若*agree*=true，則此獨立R_3同意獨佔資源使用。

end procedure

演算法 3.4 詢問進入臨界區間的使用權

Procedure request(節點 p，節點規模 n)

p 為發出詢問之節點。

步驟:

1. 呼叫getAgreement(p); 取得該R_3的同意。
2. 呼叫 compMesh (n); 計算矩陣大小 m * l。
3. 呼叫compXY(p的尾序); 計算R_3座標 (x, y)。
4. 呼叫getAgreement(k, y), $1 \le k \le m$; 依序取得同一行R_3群組的同意。
5. 呼叫getAgreement(x, k); $1 \le k \le l$; 依序取得同一列R_3群組的同意。
6. 若所有群組皆同意請求，則節點 p 獲得進入臨界區的權利。

end procedure

演算法 3.5 回覆取得獨佔資源之請求

Procedure reply(節點 p)

p 為發出詢問之節點。

初始值：*agree*=false;

步驟：

1. 判斷是否已同意他人之請求，若尚未同意，則 *agree* = true。
2. 利用最短路徑傳送 *agree* 給 p。

end procedure

以R_5為例，一個R_5共有 5 * 4 = 20 個獨立R_3，我們可以將此 20 個R_3對應至一個 5* 4 的矩陣節點中，如圖 3 所示。假設我們要計算獨立R_3 = *25(代表節點編號尾序為 25 的 6 個節點)對應於矩陣的座標(x, y)，可計算如：獨立R_3 = *25 的序數seq = (2 – 1) * (5 – 1) + (5 – 1) = 8，其對應於座標(x, y)為x= 8 – 5 * floor(8/5) = 3 且y = ceil(8/5) = 2，得出*25 位於(3, 2) 的座標上。因此當*25 欲進入臨界區間時，它必須取得同一行(*23、*24、*31 及*32)及同一列(*14、*41 及*52)的獨立R_3的同意。

*12	*23	*34	*45
*13	*24	*35	*51
*14	*25	*41	*52
*15	*31	*42	*53
*21	*32	*43	*54

圖 3　R_5之mesh對應

定理 3-3. *演算法 3.4* 可達成互斥作業。

證明：我們利用矩陣排列方式來排列節點，任一節點請求進入臨界區間時，必須取得同一行及同一列所有R_3群組的同意。因此，在尚未釋放權利的情況，不會有另一個節點取得所屬行及所屬列的R_3群組的同意，因為必定會與已請求之節點所請求之R_3群組重覆，由此可以保證互斥。在封包的傳遞上，可以以FIFO的方式傳輸，封包時間包含該節點的時間及其節點編號，並且每一個節點皆維護一個訊息佇列，其內容依封包時間排序。因此訊息不會互相等待而發生死鎖(dead lock)及或一直無法排到處理而發生餓死(starvation)。 □

定理 3-4. 若旋轉圖的節點數為 m，演算法 3.4 完成互斥詢問的法團大小約為 $2\sqrt{m/6}-1$。

證明：我們以矩陣的排列方式來排列獨立R_3，而一個R_n具有$m/6$ 個R_3。而*演算法 3.4* 的排列方式為方陣型式排列，因為只要取得同一行及同一列群組的同意，故法團大小約為 $\sqrt{m/6}+\sqrt{m/6}-1$，即 $2\sqrt{m/6}-1$。 □

3.2 以最短路徑衍生樹為基礎之互斥問題演算法

在本節中，我們利用旋轉圖的最短路徑衍生樹的特性完成另一種互斥演算法。在旋轉圖中，任二點之間的最短路徑為唯一的，所以旋轉圖也具有唯一的最短路徑衍生樹，如圖 4 所示。藉由最短路徑衍生樹的節點分布特性，我們採取多數決定的機制，在最短路徑衍生樹中拜訪一半以上的節點，以取得進入臨界區間的權利。此演算法不需前置計算，即可很快地在最短路徑衍生樹中散布請求。

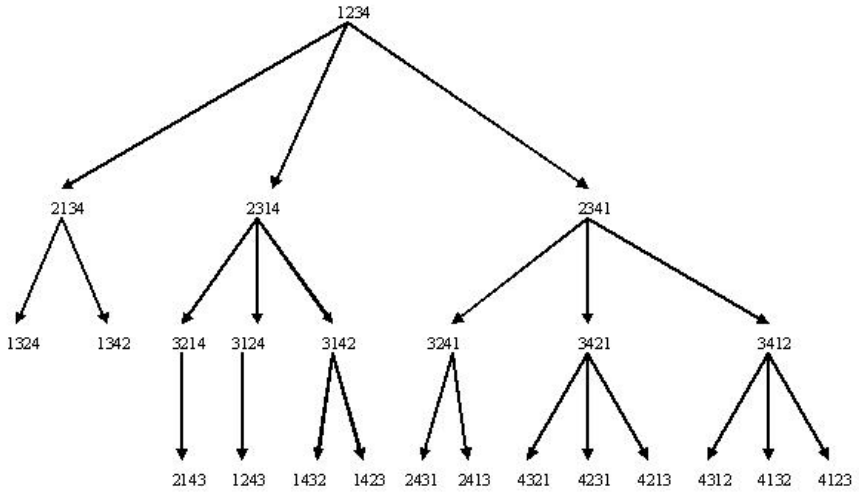

圖 4　R_4之最短路徑衍生樹

對於一個R_n而言，在樹根節點為 123…n的最短路徑衍生樹中，我們定義level[i]為節點的集合，其中從樹根到集合中節點的最短路徑為 i，在此 $1 \le i \le n-1$。此外，在此最短路徑衍

生樹中，我們亦定義子樹subTree[j]，$1 < j < n + 1$，代表由樹根經由g_j邊所連接的子樹，不同的level[]及subTree[]皆具有不相同的特性。我們由以下定理說明。

定理 3-5. 在R_n以 123…n為樹根的最短路徑衍生樹而中，假設S為level[i]中的某一節點，則 $p(i + 1, S) < p(i + 2, S) < \ldots < p(n, S)$ 且$p(i + 1, S) < p(i, S)$，$1 \le i \le n - 2$。

證明：在最短路徑衍生樹中，由起始節點到各level[i]的距離為i，因此其有序數列長度為 $n - i$。假設存在一個排列，其並未維持 $p(i + 1, S) < p(i + 2, S) < \ldots < p(n, S)$ 之排序，表示存在一個k，使得$i + 1 < k < n$ 並滿足$p(i+1, S) > p(k, S)$且$p(k, S) < p(n, S)$。則其有序數列長度$= n - k + 1 > n - i + 1 > n - i$，即此節點並非位於level[$i$]中。另假設 $p(i + 1, S) > p(i, S)$，表示 $p(i, S) < p(i + 1, S) < \ldots < p(n, S)$，則有序數列長度 $\ge n - i + 1 > n - i$，即此節點並非位於level[i]中。 □

定理 3-6. 在R_n以 123…n為樹根的最短路徑衍生樹而中，假設S為level[n-1]中的某一節點，則$p(n, S) < p(n - 1, S)$。

證明：在最短路徑衍生樹中，由起始節點到level[n-1]的距離為n-1，因此其有序數列為 1。假設$p(n, S) > p(n -1, S)$，則有序數列長度$\ge n - (n - 1) + 1 \ge 2 > 1$，即此節點並非位於level[$n$ - 1]中。 □

定理 3-7. 在R_n以 123…n為樹根的最短路徑衍生樹而言，subTree[j]中的任一節點S具有$p(1, S) < p(j + 1, S) < \ldots < p(n, S)$ 之特性，$2 \le j \le n - 1$。

證明：旋轉圖的最短繞徑方式和插入式排序法(insertion sort)相同，每個排序步驟均將起點的第一個節點編號插入正確的已排序序列間，因此每一個符號在排序過程中只會最多調整位置一次。因此當衍生樹樹根(節點 123…n)的符號 1 藉由g_j插入到符號(j+1)之前以後，即不會再調整符號 1 的位置，且符號(j+1)到n也因此無法旋轉至到第一個位置而調整位置，因此子樹subTree[j]中的節點必定會維持$p(1, S) < p(j + 1, S) < \ldots < p(n, S)$之特性。 □

定理 3-8. 在R_n以 123…n為樹根的最短路徑衍生樹而言，subTree[n]中的任一節點S具有$p(n, S) < p(1, S)$之特性。

證明：同*定理 3-7*，旋轉圖的最短繞徑方式和插入式排序法(insertion sort)相同，因此每一個符號在排序過程中只會最多調整位置一次。而當衍生樹樹根(節點 123…n)的符號 1 藉由g_n插入到符號n之後，即不會再調整符號 1 的位置，且符號n也因此無法旋轉至到第一個位置而調整位置，因此子樹subTree[n]中的節點必定會維持$p(n, S) < p(1, S)$ 之特性。 □

由以上特性我們可以發現，起始節點僅需要訪問 subTree[n]，再加上自己本身，即可取得一半以上之節點的同意。另外，由於旋轉圖是對稱的，在以下的討論中均以節點 123…n為發起詢問的節點，或稱為樹根節點。演算法如下所示：

演算法 3.6 詢問進入臨界區間的使用權

Procedure visitSubTree(樹根節點，節點規模 n)

步驟：

1. 呼叫request(節點 234..n1); /*走g_n方向，向subTree[n]的樹根發出請求*/
2. 若子樹皆同意請求，則樹根節點獲得進入臨界區間之同意。

end procedure

演算法 3.7 向某節點請求進入臨界區間

Procedure request(節點 source)

source 為收到 request 之節點本身。

步驟：

1. 判斷自己是否已同意他人進入臨界區間，若已同意，則走最短路徑回覆起始節點不同意。反之則繼續進行以下步驟。
2. 計算從樹根節點到節點 source 的最短路徑長度 k，若 $k = n - 1$，則表示已到最後一層節點，不需再往下傳遞訊息。
3. 依序判斷子節點是否符合 level[k+1]之特性，符合則傳送 request。

```
for (i = 2 to n) {
    令s為source藉由g_i外連之節點;
    若 s 符合 level[k + 1] 之節點特性則
        執行 request(s);
    }
end procedure
```

由以上說明，根據最短路徑衍生樹節點分布的特性，我們可以很快地散布訊息並收集資訊，取得一半以上節點之同意以進入臨界區間。以圖 5 為例，若節點 1234 欲進入臨界區間，僅需取得節點 2341, 3241, 3421, 3412, 2431, 2413, 4321, 4231, 4213, 4312, 4132 及 4123 的同意，再加上自己本身，即取得一半以上共 13 個節點的同意，並可進入臨界區間。

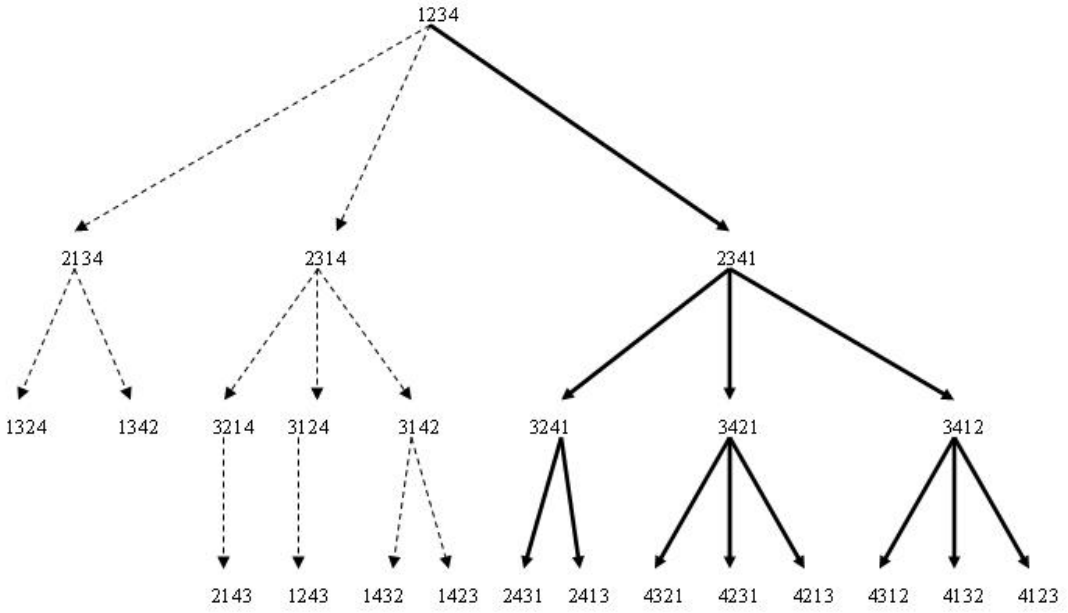

圖 5 旋轉圖R_4之詢問請求路徑

定理 3-9. *演算法 3.7* 可達成互斥作業。

證明：由 *定理 3-8* 得知 subTree[n]維持 p(n, S) < p(1, S)的節點順序，因此共有 n!/2 個節點存在於 subTree[n]中。而發出請求的起始節點必定能獲得自己的同意，因此共可獲得(n!/2) + 1 個節點之同意。若此時有另一個欲進入臨界區間的節點，則無法再獲得一半以上節點的同意，因此可以保證互斥。在封包的傳遞上，可以以 FIFO 的方式傳輸，封包時間包含該節點的時間及其節點編號，並且每一個節點皆維護一個訊息佇列，其內容依封包時間排序。因此訊息不會互相等待而發生死鎖(dead lock)及或一直無法排到處理而發生餓死(starvation)。 □

4. 演算法比較

在以矩陣為基礎的互斥問題演算法中，我們主要利用方陣結構中同一行與同一列的R_3群組，必定與另一行及另一列的R_3群組有所交集的特性，來達到保證互斥的目的，因此其詢

問的節點數量能夠大量減少。而以最短路徑衍生樹的特性所設計出來的互斥問題演算法，則是沿著最短路徑衍生樹的邊傳送詢問請求，而能夠快速的在整棵樹散布請求，並取得回應。比較上，以矩陣為基礎的互斥問題演算法，需要詢問的節點數量較少，但其需花費時間計算矩陣的對應以散布詢問請求；而以最短路徑衍生樹為基礎的互斥問題演算法，其需詢問的節點數量較多，但它可以快速地在衍生樹中散布詢問請求而不需前置計算。

5. 結論

本論文提出二種在旋轉圖中解決互斥問題的演算法。在以往的文獻及研究中，尚未有與旋轉圖相關的互斥問題演算法被提出。本論文所提出之兩種演算法，充分利用旋轉圖的拓樸特性，包括其遞迴建構、唯一最短路徑及節點編號排列的特性。本論文中，以矩陣為基礎之互斥問題演算法，其利用旋轉圖的遞迴建構特性，可以將R_n分解成許多獨立的R_3，並利用一個矩陣資料結構的節點用以對應這些獨立R_3，其優點為各節點可以快速計算出應詢問的R_3，並且詢問的R_3數量約為$2\sqrt{n!/6}$。另一個以最短路徑衍生樹為基礎之互斥問題演算法，其利用旋轉圖的唯一最短路徑及節點編號排序的特性，其優點為詢問封包可直接沿著最短路徑衍生樹的邊傳送，不需經由中間節點轉送。比較上，以矩陣為基礎之演算法詢問較少數量的節點，而以最短路徑衍生樹為基礎之演算法則不需前置計算即可在整棵樹中散布請求。

參考文獻

[1] K. Kaneko and Y. Suzuki. An algorithm for node-to-set disjoint paths problem in rotator graphs. ***IEICE Trans. Inf. & Syst***, Vol.E84-D(9): 1155 – 1163, 2001.

[2] Mamoru Maekawa. A $\sqrt{N}$ Algorithm for Mutual Exclusion in Decentralized Systems. ***ACM Transactions on Computer Systems***, Vol. 3, No2, 1985.

[3] Peter F. Corbett. Rotator graphs - an efficient topology for point-to-point multiprocessor networks. ***IEEE Transactions on parallel and Distributed Systems***, vol. 3(5):622-626, 1992.

[4] Subburajan Ponnuswamy and Vipin Chaudhary. Embedding of cycles in rotator and incomplete rotator graphs. ***Sixth IEEE Symposium on Parallel and Distributed Processing***, 1994.

[5] Subburajan Ponnuswamy and Vipin Chaudhary. Embedding of Meshes on Rotator Graphs. ***Parallel and Distributed Computing Laboratory***, Tr-93-03-03, 1993.

[6] Subburajan Ponnuswamy and Vipin Chaudhary. Low Latency Routing Algorithms for Rotator and Star Networks. ***Parallel and Distributed Computing Laboratory***, Tr-93-04-04, 1993.

[7] Y. Hamada, F. Bao, A. Mei, and Y. Igarashi. Fault-tolerant file transmission by information dispersal algorithm in rotator graphs. ***IEEE Proceedings of the 16th ICECS***, 19 – 25, 1996.

[8] Y. Hamada, F. Bao, A. Mei, and Y. Igarashi. Nonadaptive fault tolerant file transmission in rotator graphs. ***IEICE Trans. Fundamentals***, Vol.E79-A(4): 477-482, 1996.