在二進制無向性 de Bruijn 網路上的容錯繞路及廣播演算法

Fault Tolerant Routing and Broadcast Algorithms for Binary Undirected de Bruijn Networ s

石俊彬
J. B. Shi
逢甲大學資訊工程學系
Department of Information Engineering,
Feng Chia University
spooky@chinese-com.com.tw

黃秀芬
Shiow-Fen Hwang
逢甲大學資訊工程學系
Department of Information Engineering,
Feng Chia University
sfhwang@fcu.edu.tw

摘要

由於 de Bruijn 網路具有固定階度的優點，使得它成為超方體網路外，另一受歡迎的連接網路。本論文在 de Bruijn 網路上，探討當今網路上的兩大重要問題—容錯繞路和廣播。首先，在二進制無向 de Bruijn 網路（表示成 UB(2,n)）上，我們提出無訊息重複的廣播演算法，並將其推廣到 UB(r,n) 上。其次，在 UB(2,n)上提出一個容一個錯的容錯繞路演算法，此演算法所找出的路徑長度，最差比最短路徑多 4 步，而若最短路徑上無錯誤點，則必可將其找出。最後，也在 UB(2,n)上得到一個容錯廣播演算法。

Keyword *: de Bruijn networks, routing, broadcasting, fault-tolerant, inter-connection networks.*

一. 簡介

在現有的網路拓蹼中，超方體(*hypercube*)無疑是最強勢的結構，但是超方體架構具有一個很大的缺點，便是每個點的階度會隨著點個數增加而成長，這使得超方體在擴充上有很大的困難。於是，學者們尋找具固定階度之圖。而 Pradhan 和 Reddy[10] 所提出的 *de Bruijn* 圖即是其中之佼佼者。因為此圖除具有固定階度外亦有小的直徑和容錯能 力等好的性質。

雖然 *de Bruijn* 已受到學者的關注（[4], [11] 等），也被宣稱頗具實用價值，然而和超方體比起來，於其上的研究成果還是相當少，尤其在內部連接網路中容錯和繞路這兩個重要議題研究上（ [7], [9], [10]等），尚未有較好的研究成果被提出。因此，我們擬對其做探討與研究，以期更提高其實用性。

本篇論文主要的架構如下：第二節描述 *de Bruijn* 網路及其最短路徑繞路演算法；第三節提出提出一個 *UB*(2,*n*)上無訊息重複之廣播演算法，並將其推廣到 *UB*(*r*,*n*)上；第四節提出一個 *UB*(2,*n*)上的容錯繞路演算法，並將其利用到容錯廣播上；第五節結論。

二. de Bruijn 圖

2.1 定義和符號

我們將一個內部連接網路看成是 *G*(*V*個無向 *E*)，其中集合 *V* 稱為點，其對應所有處理器，集合 *E* 稱為邊，其對應處理器間雙向的通道(link)。

定義 2.1 一個無向 *de Bruijn* 圖 *UB*(*r*, *n*)是一個簡單 (simple) 圖，由 $N= r^n$ 個點組成，標示成 1,2,..., r^n，其每一個點 x 表示成 $x=(x_1, x_2,..., x_n)$，其中 $0 \le x_i \le r-1, 1 \le i \le n$。而每一個點 x 連接點 $(\alpha, x_1, x_2,..., x_{n-1})$ 和 $(x_2, x_3,..., x_n, \beta), \forall\ 0 \le \alpha, \beta \le r-1$。圖 1 是一個 *r*=2, *n*=4 的 *de Bruijn* 圖的例子。

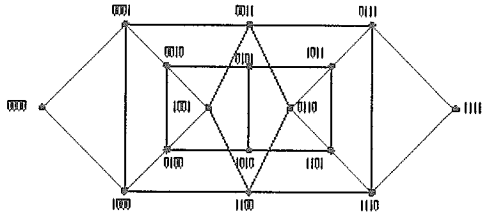

圖 1. *UB*(2, 4)

點 $(\alpha, x_1, x_2,..., x_{n-1})$ 稱為 x 的 *R*-鄰點，而 $(x_2, x_3,..., x_n, \beta)$ 稱為 x 的 *L*-鄰點，我們用 *R*(*x*)和 *L*(*x*)分別表示成點 x 的 *R*-鄰點所成的集合和 *L*-鄰點所成的集合。而左移函數 *L* 定義如下：

$$L_d(x_1x_2...x_n)=x_2x_3...x_nd$$

右位移函數定義如下：

$$R_d(x_1x_2...x_n)=dx_1x_2...x_{n-1}$$

其中 $d\in\{0,1,...,r-1\}$。

而一個(v_0,v_k)-路徑，表示一個路徑從 v_0 到 v_k，一個(v_0,v_k)-路徑 $Q=<v_0,v_1,...v_k>$ 是一個 *R*-路徑，表示 $v_i\in R(v_{i-1})$ 對於 $1\le i\le k-1$，而一個(v_0,v_k)-路徑被稱為 *L*-路徑，表示 $v_i\in L(v_{i-1})$ 對於 $1\le i\le k-1$。為了方便我們通常用 *R* 和 *L* 來分別代表一個 *R*-路徑和 *L*-路徑。並將路徑的長度表示成小寫的字母，例如：$|L_1|$表示成 l_1。而用 P_{xy} 和 $d(x, y)$分別代表 *x, y* 之間的最短路徑和最短距離。

2.2 最短路徑繞路

經由文獻[8]中 Zhen Lin 和 Ting-Yi Sun 對 $UB(r,n)$ 上最短路徑繞路的研究，可以知 $UB(r,n)$最短路徑的形態必以下面的方式存在：

Type 1: $P_{xy}=\begin{cases}R_1LR_2 & and\ |L|>\max\{|R_1|,|R_2|\}\\ L_1RL_2 & and\ |R|>\max\{|L_1|,|L_2|\}\end{cases}$

Type 2 $P_{xy}=RL\ or\ LR$

Type 3: $P_{xy}=R$ or *L*.

其可分成三種 *Type*，但每一 *Type* 中又可細分成子 *Type*，因此我們再將其細分，例如 *Type* 1 中的兩個子 *Type*，R_1LR_2 我們將其編號為 *Type* 1.1，而 L_1RL_2 編號為 *Type* 1.2。同理 *RL* 編號為 *Type* 2.1、*LR* 為 *Type* 2.2，*R* 為 *Type* 3.1，*L* 為 *Type* 3. 。圖 2 為 *Type* 1. 的例子，圖中灰色部份為共同字串。

圖 2. *Type* 1.1 R_1LR_2

由以上的三種 *Type* 可知 $UB(r,n)$的最短路徑和 *X, Y* 兩字串的最大共同字串有很大的關係，在文獻[8]中提出 Prefix tree 演算法，來找出最短路徑 *type*，為了我們的使用方便，我們將其改名為 *FSPT*(*x*,*y*)。

三. $UB(2,n)$上無訊息重複之廣播演算法

在這一小節中我們提出一個 $O(\log N)=O(n)$ 的 *one-to-all* 廣播演算法。我們將利用最短路徑來判斷是否傳送訊息，但因兩點間最短路徑可能不只一條，可能會造成訊息重複傳遞，因此我們需要對兩點間的最短路徑做分析。

經由對三種繞路形態做分析，可知何時兩點間的最短路徑將不只一條。以 *Type* 1.1 來當例子，在繞路過程中，很容易發現在 R_1 階段中往右位移的字元可為任意字元，其並不影響繞路的過程，我們將這段過程稱為 **don't care**，並以 d 代表這些字元；而在 *L* 階段的前半段其往左位移的字元就不是任意的了，其必須對應到 *y* 中 $l-r_1$ 的字元了，我們將這段過程稱為 **care**，並以 c 代表這些字元；同理，*L* 的後半段為 don't care；而在 R_2 為 care。

輔理 3.1 當 *FSPT*(*x*,*y*)[8]輸出的 *Type* 為 *Type* 1.1、*Type* 1.2、*Type* 2.1、*Type* 2.2、*Type* 2.3、*Type* 2.4 中的任一個或輸出的 *Type* 有兩個以上時，*x* 到 *y* 的最短路徑將不只一條。

證明：

很輕易的我們知 *Type* 1.1、*Type* 1.2、*Type* 2.1、*Type* 2.2、*Type* 2.3、*Type* 2. 中，其都有階段是 don't care，將任意的一個 don't care 字元設成 1，或是設成 0 可以得到兩條不一樣的最短路徑。但我們並保證這樣找到的路徑是點互斥路徑。而當輸出的 *Type* 有兩個以上時(i) 假如輸出的兩個 *Type* 中有 *Type* 1.1、*Type* 1.2、*Type* 2.1、*Type* 2.2、*Type* 2.3、*Type* 2. 中的任意一個，則最短路徑必不只一條；(ii)假如(i)不成立則其輸出 *Type* 必為 *Type* 3.1 和 *Type* 3.2，也就是單一方向往左往右皆有最短路徑，也可知 *x* 到 *y* 的最短路徑必不只一條，由(i)和(ii)故得證。 □

接著，我們定義編號最高為 *Type* 後面的編號最大的那個，即以 3.2 > 3.1 > 2.4 > 2.3 > 2.2 > 2.1 > 1.2 > 1.1 的順序來選編號。

輔理 3.2 利用選編號最高的子 *Type*，以及將 *d* 設為 1，而 *c* 設為該對應的字元，可在 *x* 到 *y* 的眾多最短路徑中選出唯一的一條最短路徑。

證明：

因為其為 *FSPT*(*x*,*y*)選出的 *Type*，故可知照其方法繞路皆為最短路徑，接著證明在眾多最短路徑中僅選出一條最短路徑，利用反證法，假設在一特定的 *Type* 下，將 *d* 設為 1，而 *c* 設為該對應的字元能造出兩條不同路徑 P_{xy}, Q_{xy}，但我們知此兩條路徑所位移的字元和方向都是一樣的，故知 $P_{xy}=Q_{xy}$ 矛盾，故得證。

□

根據上面的定理，我們將提出一個 FFP_{xy} 演算法，來找出 *x* 到 *y* 中優先權最高的最短路徑。

FFP_{xy} (*x*, *y*, *output type of FSPT*)

input: *x*, *y* 和 *FSPT* 輸出的 *type*

output: 一條我們定義最高優先權的路徑 FP_{xy}

begin

step 1.在所有 *FSPT* 輸出的 *Type* 中，選出 *Type* 編號最高的那一個。

step 2.將所選出的子 *Type*，依輔理 3.2 所描述的依

序繞路，並將繞路過程中經過的點存入 FP_{xy} 。

end

因為 *FSPT* 演算法時間複雜度為 $O(n)$，而位移字元 d 和字元 c 的次數加起來小於或等於 n，且位移的動作僅需常數時間，故可知 FFP_{xy} 演算法時間複雜度為 $O(n)$。

利用此最高優先權路徑 FP_{xy}，即可解決訊息重複傳遞問題，在許多會傳給 y 的點中，我們僅讓最高優先權路徑 FP_{sy} 上，y 的前一個點 x 傳，而其他最短路徑 y 的前一個點都不用傳，這樣就解決了訊息重複傳遞的問題了。我們將這個找出最高優先權路徑上 y 的前一點的函數定成 $node\_before\_y\_in\_FP_{sy}$，此函數僅需修改 FFP_{sy} 的輸出就可達到目的了。分析 $node\_before\_y\_in\_FP_{sy}$，因其為修改 FFP_{sy} 而來，只是輸出由原來的最高優先權路徑改為路徑上 y 的前一點，故可知 $node\_before\_y\_in\_FP_{sy}$ 的時間複雜度亦為 $O(n)$。

以下將正式提出我們的廣播演算法：

在 $UB(2,n)$ 上每個點執行以下的 *One_to_All_Broadcas* 演算法。

One_to_All_Broadcast(s, x)

Input: 起點 s 和 $UB(2,n)$ 上任一點 x

Output: $UB(2,n)$ 上每一個點都收到訊息且只收到一次

begin

1. if x 收到訊息 then

2. begin　　//開始試著傳訊給其鄰點//

3. $\forall y \in N(x)$　　// 為 x 的四個鄰點//

4.if (d(s,y)=d(s,x)+1 and node_before_y_in _ FP_{sy} =x)

5. then x 送訊息給 y.

6. end

end

觀察我們的 *One_to_All_Broadcas* 演算法，可發現有四個鄰點共做了四次 if $d(s,y)= d(s,x)+1$ and $node\_before\_y\_in\_FP_{sy}=x$ 的條件判斷，以上的函數 d 和 $node\_before\_y\_in\_FP_{sy}$ 其複雜度都是 $O(n)$，而 x 送訊息給 y 僅需常數時間，故可大約估計出 *One_to_All_Broadcas* 演算法的時間複雜度是 $O(n)$。

以下對我們演算法的正確性做驗證

定理 3.3：*One_to_All_Broadcas* 演算法確保 $UB(2,n)$ 中的所有點皆能收到訊息，且每一個點僅收到訊息一次。

證明：

對 *distance* d 做數學歸納法。

當 d=1 時：

距離 s 等於 1 的點，顯而易見的其必滿足 $d(s,y)= d(s,x)+1$。

假設 d=n-1 時成立。

則當 d=n 時：

假設任一 d=n 的點 y 首先證必有 y 的鄰點 x 使得 $d(s,y)=d(s,x)+1$，假設不存在 x 使得 $d(s,y)=d(s,x)+1$，則必存在一 P_{sy}，且 P_{sy} 沒有經過 y 的鄰點，此為矛盾，故得證必有 x 使得 $d(s,y)=d(s,x)+1$。接著證明唯一性，即證當 s 到 y 的最短路徑不只一條，必存在唯一 x 使得 $node\_before\_y\_in\_FP_{sy}=x$，由輔理 3.2 可知我們選到的 FP_{sy} 僅有一條，故知 y 的前一個點 x 必唯一。

□

而對於在整個 $UB(2,n)$ 上廣播所需的時間分析如下，由於訊息是經最短路徑傳送的，可知當將廣播的過程表示成以 s 為根的樹狀結構時，其高度為 n，因此如果是在 *all_port* 時，$UB(2,n)$ 上廣播所需的時間為 $O(n)$，而如果是在 *one_port* 時，以最差的情況來分析此廣播樹，亦即其為完整樹見圖 3

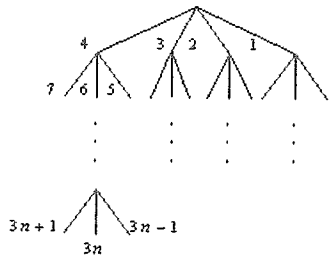

圖 3.　$UB(2,n)$ 上對應廣播的廣播樹

考慮最右邊的子樹，可知最後一個點將在 $3n+1$ 步後做 *One_to_All_Broadcas* 的動作，因此在 *one_port* 的情況下時間複雜度為 $O(3n+1)=O(n)$。

透過將 d 設成 r-1，另外將 *One_to_All_Broadcas* 演算法中鄰點的個數由 4 改為 $2r$，如此簡單的修改就可以將 $UB(2,n)$ 上 *One_to_All_Broadcas* 演算法擴充到 $UB(r,n)$ 上。

四．$UB(2,n)$ 上的容錯繞路演算法

在本小節我們將提出一個 $UB(2,n)$ 上的容錯繞路演算法，此演算法所找出的路徑長度，最差比最短路徑多 4 步，而若最短路徑上無錯誤點，則必可將其找出。

假設任一點 $X=x_1x_2...x_n$ 壞掉，則 X 的四個鄰點分別為 $L_1(X)$，$R_1(X)$，$L_0(X)$ 和 $R_0(X)$，我們定義 $L_1(X)$，$L_0(X)$ 和 $R_1(X)$，$R_0(X)$ 為同方向鄰點，而其餘組合如 $L_1(X)$ 和 $R_1(X)$ 為不同方向鄰點。

輔理 4.1　令 X 為錯誤點，則其鄰居避開 x 的繞路可分為以下兩種類型，X 的同方向鄰點間繞路，其替代路徑分別為 $x_2...x_n1 \leftrightarrow \overline{x_1}x_2...x_n \leftrightarrow x_2...x_n0$ 和 $1x_1...x_{n-1} \leftrightarrow x_1...x_{n-1}\overline{x_n} \leftrightarrow 0x_1...x_{n-1}$ 以及 X 的不同方向鄰點

間繞路，其替代路徑為 $x_2 \ldots x_n d \leftrightarrow \overline{x_1} x_2 \ldots x_n \leftrightarrow \overline{\overline{x_1} x_1} x_2 \ldots x_{n-1} \leftrightarrow \overline{x_1} x_2 \ldots x_{n-1} \overline{x_n} \leftrightarrow x_2 \ldots x_{n-1} \overline{x_n x_n} \leftrightarrow x_1 x_2 \ldots x_{n-1} \overline{x_n} \leftrightarrow d x_1 \ldots x_{n-1}$，其中 d 可為 0 或 1。

證明：

觀察以上所提的三條替代路徑，可發現路徑上經過的每一個點，其第 1 的字元或是最後一個字元，分別為 $\overline{x_1}$ 或是 $\overline{x_n}$，透過此種巧妙的安排，我們確保其一定不會經過錯誤點 x。

□

在 X 的同方向鄰點要到彼此時，容錯繞路路徑為最短路徑。而在 X 的不同方向鄰點要到彼此時，其替代路徑長度增加 4。

以下我們將定理 4.1 寫成一個演算法 *Neighbor_of_fault_communication* (x, y, f)。

Neighbor_of_fault_communication (x, y, f)。

Input: 錯誤點 f 和彼此需要通訊的兩個鄰點 x 和 y

Output: 一條 x 不經過 f 傳訊息給 y 的路徑 P

begin

Case 1. if ($x= f_2 \ldots f_n 1$ and $y= f_2 \ldots f_n 0$) or ($x= f_2 \ldots f_n 0$ and $y= f_2 \ldots f_n 1$) then

$P= f_2 \ldots f_n 1 \leftrightarrow \overline{f_1} f_2 \ldots f_n \leftrightarrow f_2 \ldots f_n 0$

Case 2. if ($x= 1 f_1 \ldots f_{n-1}$ and $y= 0 f_1 \ldots f_{n-1}$) or ($x= 0 f_1 \ldots f_{n-1}$ and $y= 1 f_1 \ldots f_{n-1}$) then

$P= 1 f_1 \ldots f_{n-1} \leftrightarrow f_1 \ldots f_{n-1} \overline{f_n} \leftrightarrow 0 f_1 \ldots f_{n-1}$

Case 3. If ($x= f_2 \ldots f_n d$ and $y= d f_1 \ldots f_{n-1}$) or ($x= d f_1 \ldots f_{n-1}$ and $y= f_2 \ldots f_n d$) then

$P= f_2 \ldots f_n t \leftrightarrow \overline{f_1} f_2 \ldots f_n \leftrightarrow \overline{f_1 f_1} f_2 \ldots f_{n-1} \leftrightarrow \overline{f_1} f_2 \ldots f_{n-1} \overline{f_n} \leftrightarrow f_2 \ldots f_{n-1} \overline{f_n f_n} \leftrightarrow f_1 f_2 \ldots f_{n-1} \overline{f_n} \leftrightarrow 1 f_1 \ldots f_{n-1}$ (其中 d 為 0 或 1)

end

Neighbor_of_fault_communicat i o 的時間複雜度是常數，因為其替代路徑皆為常數長度。

定理 4.2 在 $UB(2, n)$上做兩點間最短路徑繞路時，利用當錯誤點發生在 d 時將 d 改設成 0，而當錯誤點發生在 c 時利用立即修正原理來做繞路，必可避開錯誤點。

證明：

因為是在 $UB(2, n)$上做繞路，錯誤的點僅有一個，因此當最短路徑上有一個錯誤的點，則可確定其餘的點皆為正確的點，因此當錯誤點是發生在 don't care 階段時，其一定是因為位移某一個 d=1 所造成的，因此如果將位移的 d 改設為 0 時，必可避開錯誤點。而當錯誤點發生在 care 階段時利用定理 4.1，即可避開錯誤點。

□

以下介紹我們的 *Instant_Recover_Fault-Tolerant_Routing*$(x, y,$ *fault_node*)演算法，其主要是修改 FFP_{xy} 而來的。

Instant_Recover_Fault-Tolerant_Routing$(x, y,$ *fault_node*)

input: x, y 和 *FSPT* 輸出的 *type*

output: 一條不經過錯誤點的路徑

begin

case 1. 當在做 FFP_{xy} 過程中假設目前繞路到點 x，如果錯誤的點 f 發生在 don't care 階段時，則將原來要位移的字元 d=1，改成 d=0，接著繼續原來的繞路。

case 2. 當在做 FFP_{xy} 過程中假設目前繞路到點 x，而當錯誤點 f 發生在 care 階段時，則將原來 $x \rightarrow f \rightarrow y$，改成 *Neighbor_of_fault_ communication* (x, y, f)的輸出結果，接著繼續 y 以後的繞路。

end

分析此演算法，因為利用到 *FSPT* 其時間複雜度為 $O(n)$，又其只是在錯誤點發生時加了一些常數時間的動作故可知 *Instant_Recover_Fault-Tolerant_Routin* 演算法的複雜度為 $O(n)$。另外我們的容錯繞路僅比最短路徑多花了 4 步，也就是在最差的情況下我們的容錯路徑的長度為 n+4，而和[9]的 $n+\log_2 n+4$ 比起來我們得到了一個較佳的結果。

利用 *One_to_All_Broadcas* 演算法和找錯誤點鄰點替代路徑的方法，我們可以做到容錯廣播，當廣播傳到錯誤點時利用其兩鄰點的替代路徑將訊息繼續傳送下去。

五. 結論

在本論文中，我們針對 $UB(2, n)$網路提出了二個演算法，一、無訊息重複廣播演算法，並將其推廣到 $UB(r, n)$，二、容一個錯的繞路演算法，並將其應用到容一個錯的廣播演算法上。在本論文中我們主要是針對 $UB(2, n)$來討論的，但並未全部推廣到 $UB(r, n)$上，在未來的研究上，我們希望能將我們的結果全部擴充到 $UB(r, n)$。

*This research is supported by the Nation Science Council of R.O.C. under contact NSC88-2213-E-035-029-

參考資料

[1] J. C. Bermond and P. Fraigniaud, "Broadcasting and gossiping in de Bruijn and networks, " SIAM Journal on Computing, Vol. 23, No. 1, pp. 212-225, 1994.

[2] J. Bruck, R. Cypher, and C. T. Ho, "Fault-fault de Bruijn and shuffle-exchange networks," IEEE Trans. on Parallel and Distributed Systems, Vol. 5, No. 5, pp. 548-553, 1994.

[3] C. Calvin and D. Trystram, "Matrix Transpose

for Block Allocations on Torus and de Bruijn Network," Journal of Parallel and Distributed Computing, 34, pp 36-49, 1996.

[4] A.-H. Esfahanian and L. Hakimi, "Fault-Tolerant Routing in de Bruijn Communication Networks," IEEE Trans. Computers, vol. 34, pp. 778-788, 1985.

[5] H. Fredricksen, "A new look at the de Bruijn graph," Discrete Appl. Math. 37/38 pp.193-203,1992.

[6] E. Ganesan and D. K. Pradham, "Optimal broadcasting in binary de Bruijn networks and hyper- de Bruijn networks," Proceedings of Seventh International Parallel Processing Symposium, pp. 655-660, 1993.

[7] Ming-Bo Lin and Ming-Hong Bai, "Broadcast Algorithms for Binary Directed de Bruijn Networks," Proceedings of National Computer Symposium, pp. 18-23,1997, R.O.C.

[8] Z. Liu and Ting-Yi Sung "Routing and Transmitting Problems in de Bruijn Networks," IEEE Trans. on Computers, Vol. 45, No. 9, pp. 1056-1062, 1996.

[9] Jyh-Wen Mao and Chang-Biau Yang, "Shortest Path Routing and Fault -Tolerant Routing on de Bruijn Networks," Conference of Internationa l Computer Symposium Taiwan. R.O.C., pp. 19-26, 1996.

[10] D. K. Pradhan and S. M. Reddy, "A fault-tolerant Communication architecture for distributed systems," IEEE Trans. Computers, vol. 31,No. 9, pp. 863-870, 1982.

[11] M. A. Sridhar and C. S. Raghavendra, "Fault-Tolerant Networks Based on the de Bruijn Graph," IEEE Trans. on Computers, Vol. 40, No. 10, pp. 1167-1174, 1991.