第 19 屆行動計算研討會

以賽局理論設計 MANET 最大獨立集合自我穩定協定

A Game-theoretic Approach to Self-Stabilizing Maximal Independent Set Protocol in MANETs

黃靖軺
國立高雄大學
資訊工程學系
ian970874@gmail.com

嚴力行
國立高雄大學
資訊工程學系
lhyen@nuk.edu.tw

葉博榮
國立高雄大學
資訊工程學系
clouds1203@gmail.com

摘要

圖論中的最大獨立集合(maximal independent set)問題是從一個無向圖 G 中的節點集合 V 挑選部分的節點集合 S，需滿足在 S 當中不存在任何兩節點相鄰且 S 不為其它任一獨立集合的子集合。本篇論文透過賽局理論(game theory)尋求較佳的最大獨立集合問題解。我們接著將賽局設計轉換為在分散式系統中運作的自我穩定演算法。自我穩定(self-stabilization)的性質可以保證不論系統初始狀態為何，必定會在有限的時間內到達穩定的合法狀態，即最大獨立集合。為了使我們提出的方法能在 MANET 中有效的實作，需另外再對演算法進行修改以克服環境上的差異。模擬實驗結果顯示我們提出的演算法有較好的效能表現，比先前的方法有較多的最大獨立集合節點數以及較短的收斂時間。

關鍵詞：最大獨立集合、賽局理論、自我穩定演算法。

Abstract

Given an undirected graph $G = (V, E)$, $S \subseteq V$ is an independent set if no nodes in S are adjacent to one another. An independent set S is maximal if no proper subset of S is an independent set. Maximal independent set problem is to find such a set S. This paper proposes a solution to this problem based on game theory. We turn the solution into a self-stabilizing algorithm running in distributed systems. The self-stability property ensures that system will enter legitimate system states in limited time regardless of initial configurations. We then convert the algorithm into a protocol that runs under MANET environment. Simulation results indicate that the proposed protocol performs better than previous work in terms of independent set size and convergence time.

Keywords: *Maximal Independent Set, Game Theory, Self-Stability*.

一、簡介

在分散式系統(distributed system)中，自我穩定(self-stabilization)的分散式演算法(distributed algorithm)可以保證不論系統初始狀態為何，必定會在有限的時間內到達穩定的正確狀態[5]。自我穩定是分散式運算(distributed computing)中一種容錯(fault tolerance)的概念，即使系統運作中途發生任何的暫態錯誤(transient fault)，不需外力的介入，系統在有限的時間內必會自行修正錯誤，而返回至合法的狀態。

分散式系統中的許多問題可以轉換成圖論(graph theory)中的問題來求解[10]。如果將分散式系統中的行程(process)視為節點，行程彼此之間進行溝通的通訊連結(communication links)視為邊，則分散式系統的網路拓撲(network topology)可以構成一個無向圖(undirected graph) G(V, E)。圖論中的獨立集合(independent set)問題是從圖 G 中的節點集合 V 挑選部分的節點集合 S，需滿足在 S 當中不存在任何兩節點相鄰。而獨立集合 S 若不為其它任一獨立集合的子集合，則稱 S 為最大獨立集合(maximal independent set)。最大獨立集合有許多實際的應用，例如在無線網路中將無線節點分割成各個叢集(cluster)[1]。利用叢集架構能夠有效減少資料傳輸的次數以達到節省能源的效果。基於最大獨立集合的方式去尋找叢集的好處是可以避免叢集首(cluster head)之

間的干擾，因而使節點能夠有效的進行通訊。

不同於其它的自我穩定演算法的設計，本篇論文基於賽局理論(game theory)，提出一個自我穩定演算法來求解最大獨立集合問題。賽局理論提供一個正規的數學架構去分析個體之間複雜的互動情況，透過賽局理論尋求個體彼此利益衝突下最適合的策略解。過去幾年來，賽局理論帶給各類學科創新的發展，範圍包括工程學、經濟學、生物學、政治學和軍事戰略……等領域。一個賽局的組成要素包含參賽者(player)、策略(strategy)和利益值函數(utility function)，每位參賽者會獨自的選擇策略來盡可能最大化本身的利益值。我們會證明在本篇論文所提出的非合作賽局不論參賽者的初始策略為何，賽局必定會收斂至納許均衡(Nash equilibrium)，且結果必為最大獨立集合。

本篇論文的貢獻整理如下：我們利用賽局理論求解最大獨立集合問題，並基於此賽局提出一個可在分散式系統中運作的自我穩定演算法。此種設計架構亦可應用在其它的問題上，例如求解最小支配集合(minimal dominating set)問題[15]。除此之外，我們也考慮如何修改此自我穩定演算法，以便實作在無線隨意網路(mobile ad-hoc network; MANET)的環境中。我們在各種網路拓撲分別以不同的設定進行模擬實驗，並與其它的相關自我穩定演算法相比。實驗結果顯示我們所提出的方法在效能方面，最大獨立集合節點數較多且到達穩定的收斂時間較短。

二、背景知識與相關研究

(一) 最大獨立集合

給定一個無向圖 G(V, E)，V 為 G 中所有點的集合，E 為 G 中所有邊的集合。若選定某個節點集合 N⊆V 且 N 中的任兩節點互不相鄰，則稱 N 為獨立集合。若 N 不屬於任何其它獨立集合，則稱 N 為最大獨立集合。

(二) 自我穩定演算法

自我穩定的分散式演算法必須符合兩種性質[5]：收斂性(convergence)和封閉性(closure)。收斂性的性質是指系統從任何的起始狀態經過有限的時間後皆會收斂至合法狀態(legitimate state)；封閉性的性質是指系統收斂至合法狀態後不會再脫離合法狀態。本篇論文要求的合法狀態即為滿足最大獨立集合的定義。

分散式系統中，個別行程需透過行程間通訊(inter-process communication)取得其它行程的資訊。自我穩定演算法的相關研究通常假設共享記憶體(shared memory)或稱共享變數(shared variable)的行程間通訊模式。在此模式中，行程間通訊乃是透過共享的記憶體空間。當某個行程的狀態改變（共享變數內容變更）時，其它的行程可以藉由讀取變數內容立刻取得最新的狀態資訊。本篇論文所提及的自我穩定的分散式演算法使用共享記憶體的行程間通訊模式。在將演算法延伸至無線網路的協定時，我們會考慮訊息傳遞(message passing)的行程間通訊模式。

若分散式系統中所有的行程皆沒有任何的識別值，個別行程對於分散式演算法來說是無法區別的，則此系統稱為匿名的(anonymous)系統。反之，某些演算法的設計要求所有行程皆有獨特的識別值以資區分，以確保系統會到達穩定。這些演算法即無法在匿名的系統上執行。

不同的自我穩定演算法可能會使用不同的運算模型(execution model)。這些運算模型假設所有行程的運算動作是由系統中一個虛擬的控制程式(daemon)來統一排程。假設一個單位時間是指執行一個步驟(step)需花費的時間。控制程式決定每個單位時間內要挑選哪些行程去執行。文獻中常見的運算模型共有三種。集中式(central daemon)控制程式在每個單位時間內只會挑選唯一的某個行程來執行。分散式(distributed daemon)控制程式在每個單位時間內會從系統中挑選部分的行程(數目介於 1 到全部的行程之間)來執行。同步式(synchronous daemon)控制程式則會選取系統中全部的行程在每個單位時間內同時執行。為保持公平性，控制程式必須確保每個行程最終一定會被挑選到。

Dijkstra 提出使用 guarded commands 的形式來表示自我穩定演算法[6]。此表示法中，自我穩定演算法由每個行程的變數(variables)和規則(rules)組成(某些文獻中，規則以動作(actions)表示)。擁有變數的行程可以改變其變數值，其他行程則只能讀取該變數值。變數值的範圍為事先定義的定義域(domain)。一條規則的形式為：＜guard＞→＜statement＞。guard 為此規則的條件，寫成與變數值相關的布林運算式(Boolean expression)。statement 為此規則的動作(action)，通常會改變一個或多個變數值。當某條規則中的 guard 為真時，稱此規則為致能的(enabled)。當某個行程有致能的規則時，稱此行程有特權(privilege)。唯有特權的

行程可以被控制程式挑選，被挑選的行程會執行其 statement。若某個行程同時有多條規則為致能的，則只有其中某一條規則的 statement 會被執行。系統的狀態由系統中所有行程的變數值組成。執行 statement 會更新某些變數的值，將系統從原先的系統狀態轉換至另一個系統狀態。若在某個系統狀態下，所有的行程皆沒有任何一條規則為致能的，則該系統狀態稱為定點(fixed point) [8]。在此種情況下我們稱該系統到達穩定狀態。

(三) 最大獨立集合問題相關自我穩定演算法

第一個求解最大獨立集合問題的自我穩定演算法由 Shukla 等人[12]提出。此演算法假設系統集中式控制程式運且系統為匿名的。每個節點需維護一個布林變數，其值表示該節點是否在最大獨立集合中。每個節點執行以下兩條規則：(1)若節點本身及其所有鄰近節點皆不在最大獨立集合中，則將變數值設為真。(2)若節點本身及其任一個鄰近節點皆在最大獨立集合中，則將變數值設為假。當系統到達穩定狀態後，變數值為真的節點集合即為最大獨立集合。

Shukla 等人的演算法假設節點沒有識別值，因此如果在分散式控制程式下運作會有無法打破鄰近兩節點間對稱性的問題。對稱性問題表現在兩個相鄰節點可能會同時執行規則(1)而加入最大獨立集合，接下來又同時執行規則(2)而離開，如此循環不已，使得系統無法達到穩定狀態。Ikeda 等人[11]因此提出一個能在分散式控制程式下運作的自我穩定演算法。此演算法要求每個節點需給定一個不與其它節點重複的獨特識別值，並將上述 Shukla 等人演算法的規則(2)修改為若節點本身及其任一個識別值較小的鄰近節點皆在最大獨立集合中，則將變數值設為假。執行此演算法時，相鄰的節點可能會同時執行規則(1)而加入集合，但不可能因滿足規則(2)的條件而同時離開集合，故可以解決 Shukla 等人演算法的對稱性問題。

Turau [13]提出一個時間複雜度小於 Ikeda 等人的自我穩定演算法，假設系統為非匿名的且在分散式控制程式下運作。每個節點除了個別的識別值外，尚需維護一個狀態值。此狀態值為 IN 表示節點在最大獨立集合中、值為 WAIT 表示想要加入最大獨立集合、值為 OUT 則表示節點不在最大獨立集合中。該自我穩定演算法包含以下四條規則：(1)若本身狀態值為 OUT 且鄰近節點的狀態值皆不為 IN，則將自身狀態值設為 WAIT。(2)若本身狀態值為 WAIT 且不存在任何識別值較小的鄰近節點狀態值為 WAIT，則將自身狀態值設為 IN。(3)若本身狀態值為 WAIT 且存在任一個識別值較小的鄰近節點狀態值為 WAIT，則將自身狀態值設為 OUT。(4)若本身狀態值為 IN 且存在任一個鄰近節點狀態值為 IN，則將自身狀態值設為 OUT。

Goddard 等人[9]提出一個在同步式控制程式下運作的自我穩定演算法。此演算法假設非匿名的系統。每個節點執行以下兩條規則：(1)若節點本身不在最大獨立集合中且也沒有識別值較大的鄰近節點在最大獨立集合中，則加入集合。(2)若節點本身以及任一個識別值較大的鄰近節點皆在集合中，則離開集合。

(四) 賽局理論

賽局理論提供了一套系統化的數理分析方法，其應用發展源自於經濟學，近年來在各個領域皆有廣泛的應用。賽局理論主要區分為「合作賽局」與「非合作賽局」兩大類。由於非合作賽局的情境較符合我們的需求，因此本篇論文著重在非合作賽局的部分。非合作賽局中，所有做決策的個體都是自私的，只會選擇能使本身獲利的策略，不會為了其它個體的利益而選擇合作。參賽者的利益可能會受到他人的決策所影響，因此非合作賽局的重點是分析與探討參賽者之間的策略性互動。

在參賽者有兩個以上的非合作賽局中，某些策略組態稱為納許均衡(Nash equilibrium)，又稱非合作賽局均衡。在這些策略組態中，其它參賽者決策不變的前提下，沒有任何一個參賽者能透過獨自的改變決策來獲得更高的利益。一般狀況下，參賽者會為了增進本身獲利而改變決策。如果賽局目前的策略組態並非納許平衡，則表示至少有一位參賽者有動機改變決策，所以目前的策略組態並非賽局的最終穩定狀態。如果此種狀況一直持續下去，則此賽局將得不到穩定解。因此賽局中是否存在納許均衡便是一個極為重要的課題。納許均衡在賽局中不一定唯一，不同均衡的情況可能會導致參賽者獲得不同的利益。另外納許均衡強調的並不是像系統最佳解所追求的整體利益最大化，而是每位獨立的參賽者皆能接受的一種平衡結果。

三、最大獨立集合自我穩定演算法

(一) 最大獨立集合賽局

我們定義無向圖中的節點為賽局中的參賽者。每位參賽者 p_i 的策略集合 S_i 為選擇加入或離開最大獨立集合。給定一策略組態

$C=\{c_1,c_2,\ldots,c_i,\ldots,c_n\}$，其中 c_i 為 p_i 的決策值，0 表示離開最大獨立集合，1 表示加入最大獨立集合。我們將參賽者 p_i 的利益函數定義如(1)所示。N_i 為與 p_i 相距一步的鄰近參賽者所組成的集合，d_i 為 p_i 的分支度(N_i 的數量)。此賽局利益函數的設計目標是盡可能增加最大獨立集合的節點總數。我們的想法是，若分支度越大的節點加入最大獨立集合，則會有越多相鄰的節點無法加入，進而減少最大獨立集合的節點總數。因此在設計利益函數時，除了讓結果滿足最大獨立集合的定義外，必須考量參賽者的分支度，希望將分支度較小的節點盡量選進最大獨立集合中。

$$u_i(C)=\begin{cases}0 & \text{if } c_i=0\\ -1 & \text{if } c_i=1\wedge\exists p_j\in N_i: c_j=1\wedge d_j\le d_i\\ 1 & \text{otherwise}\end{cases} \quad (1)$$

Theorem 1 ***定義*** $N^{-}_i=\{p_j\in N_i \mid d_j<d_i, p_i\in P\}$ ***以及*** $D_k=\{p_i \mid p_i\in P\wedge d_i=k\}$***。當*** $\bigcup_{p_i\in D_k} N^{-}_i$ ***集合的參賽者皆達到穩定或是為空集合時，則*** $\forall p_i\in D_k$ ***終究會達到穩定。***

證明：定義分支度與 i 相等且與 i 相鄰的參賽者集合為 $N^{=}_i$。當 N^{-}_i 集合的參賽者皆達到穩定時，參賽者 i 存在三種情況：(1) N^{-}_i 中的參賽者存在一個以上其決策為 1，此時對 i 最有利的策略為 0。又 N^{-}_i 的節點皆不再改變決策，因此不論 $N^{=}_i$ 中的參賽者策略為何，i 在選擇 0 後不會再改變決策，i 可到達穩定。(2) N^{-}_i 和 $N^{=}_i$ 中所有參賽者決策皆為 0，此時對 i 最有利的策略為 1。當 i 選擇 1 後，$N^{=}_i$ 中任一位參賽者改變決策並無法獲得較好的利益函數值，且 N^{-}_i 的節點皆不再改變決策，故 i 不會再改變決策，i 亦可到達穩定。(3) N^{-}_i 中所有參賽者決策皆為 0，但 $N^{=}_i$ 中至少有一參賽者決策為 1，此時對 i 最有利的策略為 0。當 i 選擇 0 後，若 $N^{=}_i$ 中有任一位參賽者 j 符合情況(2)，則 j 將策略改為 1 後可到達穩定。此時 i 的策略維持 0 會有較高的利益值，因此 i 不會再改變決策。相反的，若 $N^{=}_i$ 中所有決策為 1 的參賽者皆將策略由 1 改為 0，則 i 的情況同(2)，故 i 選擇 1 後不會再改變決策。因此情況(3)下 i 亦會到達穩定。若 N^{-}_i 集合為空集合時，則等同於情況(2)或(3)。因為 N^{-}_i 集合不會影響參賽者 i 的利益。由以上所述可得證 *Theorem 1*。

Theorem 2 ***賽局最終會達到納許均衡。***

證明：將參賽者根據分支度值以遞增的方式區分成 $D_i, D_{i+1},\ldots, D_m$，i 和 m 分別為賽局中分支度最小和最大的參賽者其分支度值。由於 D_i 為分支度最小的集合，D_i 中每位參賽者不存在任何分支度更小的鄰居。根據 *Theorem 1*，D_i 中的所有參賽者必達到穩定。因此同樣根據 *Theorem 1*，D_{i+1} 中的所有玩家必達到穩定。當此情況延伸到 $D_i, D_{i+1},\ldots, D_{m-1}$ 皆達到穩定時，D_m 中所有參賽者也必達到穩定，此時賽局中的所有參賽者不會再改變決策，賽局達到納許均衡。

Theorem 3 ***賽局達到納許均衡的結果必為最大獨立集合。***

證明：假設賽局達到納許均衡，但結果不為最大獨立集合。依最大獨立集合的定義可推論出兩種情況：(1)存在相鄰的參賽者決策皆為 1 (加入最大獨立集合)，則其中至少有一個參賽者 i 的利益函數值為-1。i 將策略改為 0 後可以獲得更高的利益函數值 0，違反納許均衡的定義，因此與假設矛盾。(2)存在一個參賽者 i 決策為 0 且相鄰的參賽者決策皆為 0。i 將策略改為 1 後可以獲得更高的利益函數值 1，違反納許均衡的定義，因此亦與假設矛盾。得證 *Theorem 3*。

(二) 最大獨立集合自我穩定演算法

我們以 guarded commands 的形式來將賽局設計轉換為自我穩定演算法。假設在分散式系統中，每個節點 i 都會有一個唯一的識別值(非匿名的系統)、固定的分支度值 d_i 和定義域為 $\{IN, OUT\}$ 的變數 status。定義 $i.id$ 代表節點 i 的識別值、$i.status=IN$ 代表節點 i 加入最大獨立集合以及 $i.status=OUT$ 代表節點 i 離開最大獨立集合。p_i 在此處指的是行程 i 或節點 i，其他符號的定義如先前所設。系統在共享變數的行程間通訊模式和同步式控制程式下執行。每個節點可以直接讀取相鄰節點的 *status* 值和 *id* 以判斷哪條規則是致能的。執行致能的規則後，節點會更新本身的 *status* 值，但無法更新相鄰節點的 *status* 值。此演算法由下列三條規則所組成：

- Rule 1

$$\exists p_j\in N_i: j.status=IN\wedge d_j<d_i \wedge i.status\neq OUT\rightarrow i.status:=OUT$$

- Rule 2

$$\exists p_j \in N_i : j.status = IN \wedge d_j = d_i \wedge j.id < i.id \wedge i.status \neq OUT \rightarrow i.status := OUT$$

■ Rule 3

$$\forall p_j \in N_i : j.status = IN \wedge d_j > d_i \wedge i.status \neq IN \rightarrow i.status := IN$$

我們透過 Rule 1、Rule 2 和 Rule 3 共三條規則將賽局設計的利益函數(1)轉換為自我穩定演算法。更新節點可能會發生暫態錯誤的 *status* 值，確保系統到達穩定狀態的結果為最大獨立集合。若在 Rule 2 中不考慮 id 大小的比較且演算法在集中式控制程式下執行，則系統的運作模式如同我們先前設計的賽局。根據 *Theorem 3*，此演算法必定會自我穩定。因為納許均衡解可以對應到系統的穩定狀態。一個是賽局中沒有任何一個參賽者因為可獲得更高的利益而改變決策，另一個是系統中沒有任何一個行程因為有致能的規則而更新變數。為了讓此演算法在同步式控制程式下執行也能保證自我穩定的性質，需在 Rule 2 中加入 *id* 的判斷以打破對稱性問題，避免分支度相同的相鄰節點同時加入最大獨立集合後又同時離開的循環情況發生。

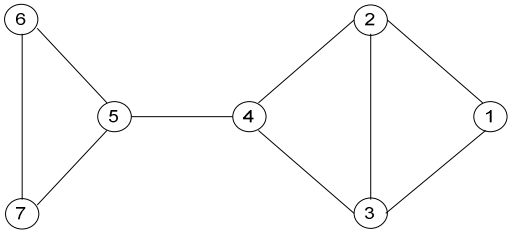

圖 1 演算法運作範例圖形

一個演算法的運作範例如圖 1 所示。圖中節點內標示的數字代表其 *id*。表 1 描述了在圖 1 中每個節點的 *status* 值在每個回合的變化情形(行代表回合，列代表節點的 *id*)。表格中括號內的 R1、R2 和 R3 代表節點在該回合執行的規則。給定一個初始狀態如表 1 中的 Initial。在回合 3 時，沒有任何節點的規則會被致能，此時系統到達穩定狀態。節點 1、4 和 6 共三個節點組成了最大獨立集合。

表 1 演算法運作情況

	Initial	1	2	3
1	OUT(R3)	IN	IN	IN
2	IN	IN(R1)	OUT	OUT
3	IN(R2)	OUT	OUT	OUT
4	OUT	OUT	OUT(R3)	IN
5	IN	IN(R1)	OUT	OUT
6	OUT(R3)	IN	IN	IN
7	OUT(R3)	IN(R2)	OUT	OUT

(三) 在無線網路的實作

如同 Demirbas 等人[4]和 Yen 等人[16]將自我穩定演算法實作在無線網路環境中會遇到許多問題。我們也必須克服無線網路環境與分散式系統的差異。以下探討將自我穩定演算法實作在無線網路環境中會遇到的幾個挑戰：

■ 行程間通訊模式的差異

自我穩定演算法通常假設程序間以共享變數的行程間通訊模式互相溝通。當有節點更新其區域變數後，其它節點後續對此變數讀取即可獲得更新後的值。無線網路中的節點並無法共享變數，意即節點內部的變數值無法讓其他鄰居節點直接讀取。變數值的任何更新須透過訊息傳遞的行程間通訊模式來通知其它節點，因此其他節點則需另行儲存此資訊。

■ 封包的遺失與延遲

無線網路中造成封包遺失或錯誤的原因很多。可能是傳送封包的訊號衰減、傳送頻道(channel)受外界干擾、傳輸碰撞、或網路擁塞時佇列(queue)緩衝空間導致將部分封包丟棄。如果無線網路的媒體擷取控制 (media access control)協定會重送遺失或錯誤的資料，則封包遺失或錯誤所造成的影響為增加封包的延遲。封包延遲的效應為節點區域變數目前的實際值可能與其他節點所知道的此變數的值不相同。如果無線網路的媒體擷取控制協定不會重送遺失或錯誤的資料，或重送不保證最後會成功，則演算法必須處理封包遺失的問題。

■ 動態的無線節點

在無線網路中除了節點變數值可能會在任何時間出錯之外，也可能因為硬體方面的問題(例如電源耗盡或是故障)而造成節點失效。另外也會有新的節點加入或離開網路的情況發生。因此與賽局或是分散式系統不同，節點的分支度可能是動態變化的。

由上述的論點可知在無線網路的環境中要面對的問題又複雜許多。因為行程間通訊模式的差異，我們為每個節點額外增加兩個陣列變數 *Cstatus* 和 *Cd*，分別用以暫存相鄰節點的 *status* 值和分支度值。定義 $i.Cstatus(j)$ 為節點 j 的 status 值在節點 i 的快取(cached)版本，

$i.Cd(j)$ 為 j 的分支度值在節點 i 的快取版本。因為無線節點的存在為動態的，所以節點必須週期性的廣播 hello 訊息，讓鄰近的節點確認該節點的存在。為此我們假設每個節點會設置一個長度固定為 T_s 的計時器，當此計時器過期時會廣播 hello 訊息並重置計時器。我們假設訊息傳遞的延遲時間上限為 λ。為了將最大獨立集合自我穩定演算法實作在無線網路，我們修改先前提出的三條規則並且額外加入三條規則如下：

■ Rule 1

$$\exists p_j \in N_i : i.Cstatus(j) = IN \wedge i.Cd(j) < d_i \wedge i.status \neq OUT \rightarrow multicast\{i.id, d_i, i.status := OUT\}$$

■ Rule 2

$$\exists p_j \in N_i : i.Cstatus(j) = IN \wedge i.Cd(j) = d_i \wedge j.id < i.id \wedge i.status \neq OUT \rightarrow multicast\{i.id, d_i, i.status := OUT\}$$

■ Rule 3

$$\forall p_j \in N_i : i.Cstatus(j) = IN \wedge i.Cd(j) > d_i \wedge i.status \neq IN \rightarrow multicast\{i.id, d_i, i.status := IN\}$$

■ Rule 4

$$\exists p_j \in N_i : rcv(\{j.id, d_j, j.status\}) \wedge i.Cstatus(j) \neq j.status \rightarrow i.Cstatus(j) := j.status \wedge i.Cd(j) := d_j$$

■ Rule 5

$$\exists p_j \in N_i : timeout(rcv(\{j.id, d_j, j.status\})) \rightarrow N_i := N_i \setminus \{p_j\} \wedge d_i := d_i - 1$$

■ Rule 6

$$\exists p_j \notin N_i : rcv(\{j.id, d_j, j.status\}) \rightarrow N_i := N_i \cup \{p_j\} \wedge d_i := d_i + 1$$

Rule 1、Rule 2 和 Rule 3 被修改成節點以快取版本的 *status* 值和分支度值來更新節點本身的 *status* 值。為了通知所有的相鄰節點，節點會利用多播(multicast)的方式傳遞本身的 *id*、更新後的 *status* 值及對應的分支度值。Rule 4 是當收到任何相鄰節點的訊息時，若與暫存的 status 值不一致，則會更新快取版本的 status 值和對應的分支度值。Rule 5 為每個節點設置一個接收訊息的計時器，時間長度固定為 T_r。若在計時器過期前沒有收到某個鄰近節點 j 發送的訊息，則判斷節點 j 已經離開網路或失效，故將 j 自 N_i 中移除。節點 i 本身的分支度值會受到影響而改變，將本身的分支度值減 1。Rule 6 是當收到某個不屬於原本鄰居的節點 j 所發送的訊息時，則判斷節點 j 為新加入網路中的成員並將 j 加入 N_i 中。節點 i 本身的分支度值會受到影響而改變，將本身的分支度值加 1。

之前的演算法設計假設當沒有任何節點的規則會被致能時，代表系統到達穩定狀態，而穩定狀態下的結果必為最大獨立集合(合法狀態)。但在無線網路的環境中，當沒有任何節點的規則會被致能時，並不保證為最大獨立集合。考慮一個簡單的範例。如圖 2 所示，節點 1 和節點 2 的 *status* 初始值皆為 OUT 且彼此皆知道對方的 *status*。此時 Rule 3 對兩個節點而言皆會被致能，因此雙方將 status 值改為 IN 並且分別多播訊息 Msg_1 和 Msg_2。因訊息傳遞會有延遲，在訊息送達前，節點 1 和 2 皆不知道對方的 status 值已從 OUT 更新為 IN。此時兩節點皆沒有任何一條規則會被致能。系統此時已到達穩定狀態，但結果卻明顯非最大獨立集合。

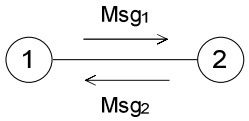

圖 2 無線網路中的延遲現象

上述的例子顯示在無線網路環境中須加入額外的假設，並重新定義系統的穩定狀態，以納入訊息延遲現象所造成的影響。如 *Definition 1* 所示，我們假設節點在 Δ 的時間內必定可以成功接收到鄰居傳送的訊息，確認有無任何其他的節點更新、失效、加入或離開網路等狀況發生。則穩定狀態可以定義成在長度為 Δ 的一段時間內沒有任何一條規則為致能的。

Definition 1 假設網路中任何節點必定能在有限的時間 Δ ($\Delta \geq T_s + \lambda$) 內成功多播訊息給所有相鄰節點。若所有的節點在某個時間點 t 到 $(t+\Delta)$ 的這段時間內沒有任何一條規則為致能的，則表示系統在 t 時到達穩定狀態。

四、模擬實驗與結果

為了研究我們提出的方法能否帶來效益，本章會針對最大獨立集合自我穩定演算法(以 Game 表示)和最大獨立集合自我穩定演算法的無線網路版本(以 Game-wireless 表示)進行模擬實驗。我們將 Game 與其它相關的自我穩定演算法 Shukla [12]、Ikeda [11]、Goddard [13]和 Turau [9]進行比較。效能評比的依據有兩個，第一個為系統到達穩定狀態時，最大獨

立集合的節點數量多寡，節點數越多則表示效能越好。第二個為系統到達穩定狀態的收斂時間。由於使用不同控制程式的自我穩定演算法無法直接比較收斂時間，因此我們只與同樣為同步式控制程式的 Goddard 進行收斂時間的效能評比(以回合為單位，所有節點會在同一回合內被挑選到)。自我穩定演算法強調不論系統初始狀態為何，系統必定會在有限的時間內到達穩定狀態，因此我們將節點的初始變數值隨機設定(加入或離開最大獨立集合)。若演算法適用於非匿名的系統，則節點的識別值同樣也是隨機設定，但識別值須唯一。所有的實驗都會輸入 1000 筆網路拓撲後，計算效能的平均值。由於求解最大獨立集合問題的結果會與網路拓撲相關，因此我們的模擬實驗測試了各種不同的網路拓撲，包含 Unit Disk Graph [3]、ER model [7]、WS model [14]、BA model [2]。

(一) 在 UDG 的實驗

UDG 的參數設定在 1000x1000 平方公尺的區域內，每個節點的訊號傳輸距離皆設定為 200 公尺。模擬實驗首先測試節點總數對效能產生的影響。實驗結果如圖 3 所示。Game 的效能優於其它四種自我穩定演算法。在 Game 和 Goddard 的收斂時間比較部分，UDG 的參數設定同上，實驗結果如圖 4 所示。Goddard 到達穩定的平均回合數在不同的節點數設定下皆比 Game 來得多。

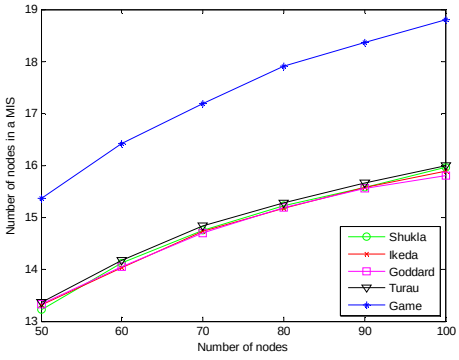

圖 3 在 UDG 的平均最大獨立集合節點數

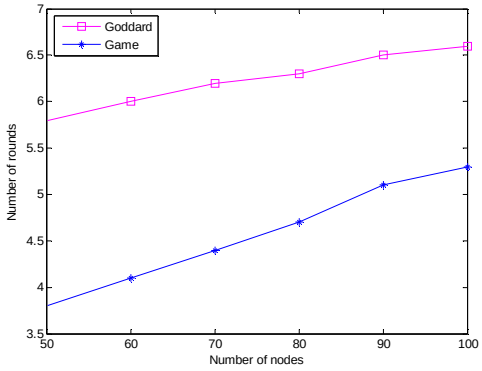

圖 4 在 UDG 的平均收斂時間

(二) 在 ER model 的實驗

我們固定節點數為 100 個，以測試任兩節點之間形成連結的機率 p_e 值對效能產生的影響。實驗結果如圖 5 所示。Game 的效能在不同的 p_e 值時仍然是最佳的。在 Game 和 Goddard 的收斂時間比較部分，ER model 的參數設定同上。實驗結果如圖 6 所示。在不同的 p_e 值時，Goddard 到達穩定所需的平均回合數皆比 Game 來得多。

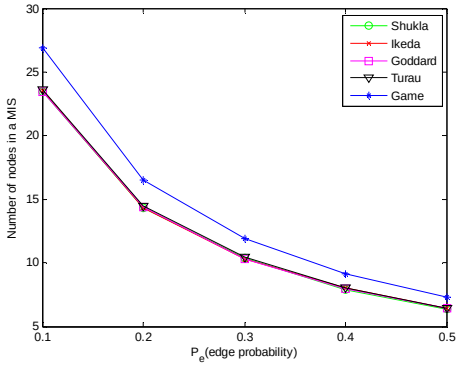

圖 5 在 ER model 的平均最大獨立集合節點數

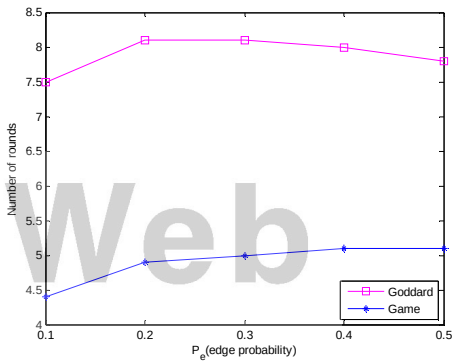

圖 6 在 ER model 的平均收斂時間

(三) 在 WS model 的實驗

我們固定節點數為 100 個，以測試邊重連機率 p_r 值對效能產生的影響。實驗結果如圖 7 所示。在 p_r 值為 0 到約 0.2 時(網路拓撲接近於規則網路)，Game 的效能較 Turau 差。因為所有節點的分支度值近乎相同，以考量分支度為解的 Game 無法發揮其效益。在 p_r 值約為 0.2 到 0.8 時，Game 會成為最佳的演算法。在 Game 和 Goddard 的收斂時間比較部分，WS model 的參數設定同上。實驗結果如圖 8 所示。在不同的 p_r 值時，Goddard 到達穩定所需的平均回合數皆比 Game 來得多。

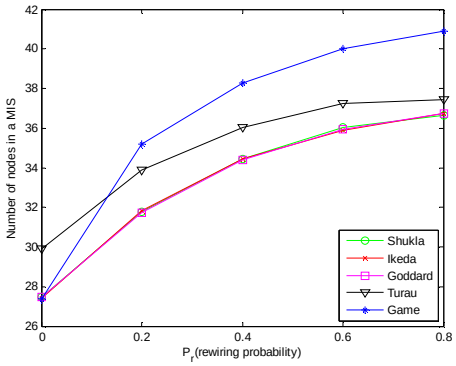

圖 7 在 WS model 的平均最大獨立集合節點數

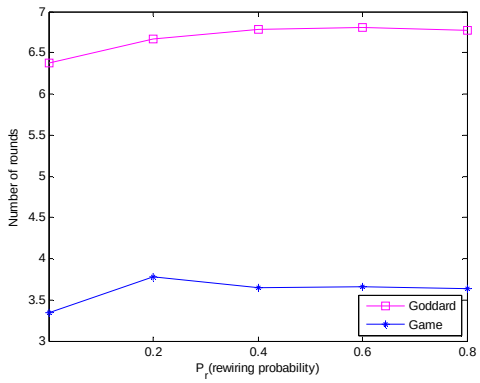

圖 8 在 WS model 的平均收斂時間

(四) 在 BA model 的實驗

我們固定節點數為 100 個，以測試新節點的邊數 m 值對效能產生的影響。實驗結果如圖 9 所示。Game 的效能在不同的 m 值時皆是最佳的。在 Game 和 Goddard 的收斂時間比較部分，BA model 的參數設定同上。實驗結果如圖 10 所示。在不同的 m 值時，Goddard 到達穩定所需的平均回合數皆比 Game 來得多。

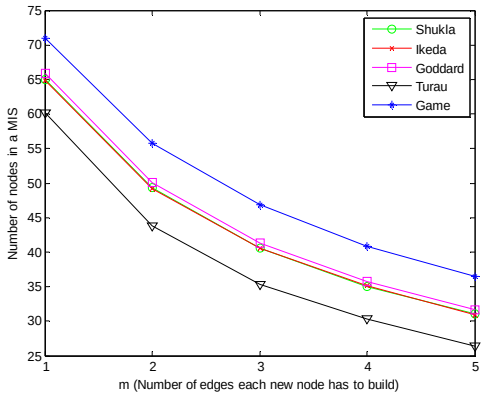

圖 9 在 BA model 的平均最大獨立集合節點數

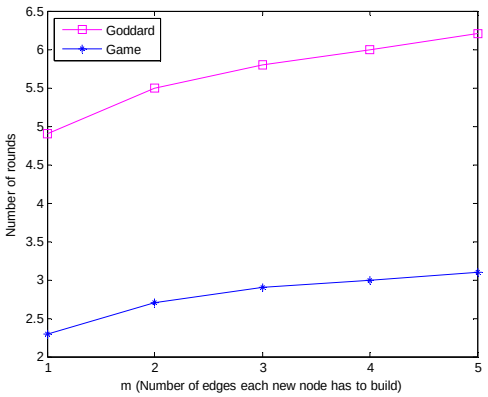

圖 10 在 BA model 的平均收斂時間

(五) 在無線網路的實驗

我們選擇常被用來模擬無線網路的 UDG 模型進行 Game-wireless 的實驗。UDG 的拓撲設定同上。Game-wireless 的設定為 $\Delta = 25$ 和 $T_s = 5$。節點運作的起始時間從 0 到 3 中隨機挑選。無線網路中的節點數量固定為 50 個，並且假設沒有節點加入或離開的情況。首先測試改變延遲時間上限 λ 值對演算法效能產生的影響，實驗結果如圖 11 所示。λ 值越大會使得節點任何的更新需花費越多的時間來通知其相鄰的節點。由於網路拓撲沒有任何變化，最大獨立集合節點數量維持在約 15 的位置，不會受到 λ 值改變的影響。

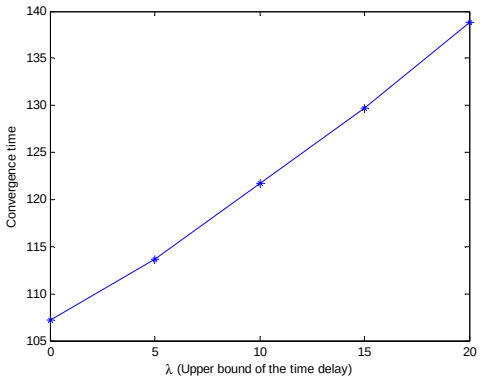

圖 11 在無線網路的平均收斂時間(改變 λ)

第二部分測試的是當無線網路中有節點失效時對演算法效能產生的影響。假設在每個單位時間皆會有機率 P 指定一個隨機的節點為失效，且失效的節點不會再啟動。設定為 $T_r = 10$ 和 $\lambda = 5$，實驗結果如圖 12 與圖 13 所示。P 值越大時，收斂時間會越長。因為演算法運作的過程中隨時可能會有節點失效造成影響，且失效的情形因為延遲的關係並無法立刻得知。最大獨立集合節點數量則會明顯隨著 P 值增加而下降。

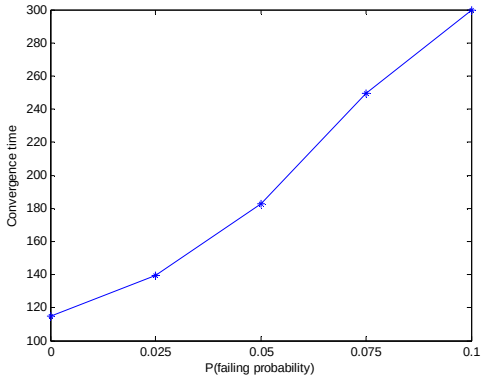

圖 12 在無線網路的平均收斂時間(改變 P)

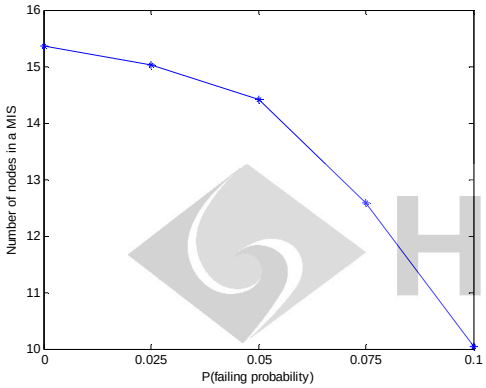

圖 13 在無線網路的平均最大獨立集合節點數

五、結論

我們提出一個基於賽局理論的自我穩定演算法，目的是希望可以在分散式系統中有效的求解最大獨立集合問題。因此我們設計了一個最大獨立集合賽局，並證明不論參賽者的初始策略為何，此賽局必定存在納許均衡。接著將此賽局設計轉換為在同步式控制程式下運作的最大獨立集合自我穩定演算法。然而，此演算法並無法直接實作在無線網路的環境中。因此我們針對這些問題進一步修改自我穩定演算法。

我們透過模擬實驗來比較 Game(我們提出的方法)與其它相關演算法的效能差異。實驗分別在 UDG、ER model、BA model 以及 WS model 等四種代表性的網路拓撲中進行。實驗結果顯示，Game 在各種網路拓撲的整體效能皆明顯優於其它的演算法。最大獨立集合的節點數較多且到達穩定的收斂時間較短。

參考文獻

[1] K. Alzoubi, P.-J. Wan, and O. Frieder, “Maximal Independent Set, Weakly-Connected Dominating Set, and Induced Spanners in Wireless Ad Hoc Networks,” *International Journal of Foundations of Computer Science*, vol. 14, no. 2, pp. 287–303, 2003.

[2] A. L. Barabási, and R. Albert, “Emergence of scaling in random networks,” *Science*, vol. 286, no. 5439, pp. 509 – 512, 1999.

[3] B.N. Clark, C.J. Colbourn, and D.S. Johnson, “Unit Disk Graphs,” *Discrete Math.*, vol. 86, pp. 165 – 177, 1990.

[4] Murat Demirbas, Anish Arora, Vineet Mittal, and Vinodkrishnan Kulathumani, “A Fault-Local Self-Stabilizing Clustering Service for Wireless Ad Hoc Networks,” *IEEE Transactions On Parallel And Distributed Systems*, vol. 17, no. 9, 2006.

[5] E. W. Dijkstra, “Self-stabilizing systems in spite of distributed control,” *Comm.ACM*, vol. 17, no. 11, pp. 643 – 644, 1974.

[6] E. W. Dijkstra, “Guarded commands, nondeterminacy, and formal derivation of programs,” *Comm. ACM*, vol. 18, no. 8, pp. 453 – 457, 1975.

[7] P. Erdos and A. Renyi, “On random graphs I,” *Publications Mathematicae, Debrecen*, vol. 6, pp. 290–297, 1959.

[8] Mohamed G. Gouda, Hrishikesh B. Acharya, “Nash equilibria in stabilizing systems,”

Theoretical Computer Science, vol. 412, no. 33, pp. 4325–4335, 2011.

[9] W. Goddard, S.T. Hedetniemi, D.P. Jacobs, P.K. Srimani, "Self-stabilizing protocols for maximal matching and maximal independent sets for ad hoc networks," *Proc. International Parallel and Distributed Processing Symposium*, 2003.

[10] N. Guellati, H. Kheddouci, "A survey on self-stabilizing algorithms for independence, domination, coloring, and matching in graphs," *Journal of Parallel and Distributed Computing*, vol. 70, no. 4, pp. 406 – 415 , 2010.

[11] M. Ikeda, S. Kamei, H. Kakugawa, "A space-optimal self-stabilizing algorithm for the maximal independent set problem," *Proc. 3rd International Conference on Parallel and Distributed Computing, Applications and Technologies*, 2002.

[12] S.K. Shukla, D.J. Rosenkrantz, S.S. Ravi, "Observations on self-stabilizing graph algorithms for anonymous networks," *Proc. 2nd Workshop on Self-Stabilizing Systems*, 1995.

[13] V. Turau, "Linear self-stabilizing algorithms for the independent and dominating set problems using an unfair distributed scheduler," *Information Processing Letters*, vol. 103, no. 3, pp. 88 – 93, 2007.

[14] D. J. Watts, and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, pp. 440 – 442, 1998.

[15] Li-Hsing Yen, Zong-Long Chen, "Game-theoretic approach to self-stabilizing distributed formation of minimal multi-dominating sets," *IEEE Transactions on Parallel and Distributed Systems*, in press.

[16] Li-Hsing Yen, Zong-Long Chen, "Self-Stabilizing Distributed Formation of Minimal *k*-Dominating Sets in Mobile Ad Hoc Networks," *The Fifth International Workshop on Wireless Network Algorithm and Theory*, 2014.