國立交通大學

電信工程研究所

碩士論文

數據中心網路之基於基因演算法的能源考量流量排程

Energy Aware Flow Scheduling for Data Center Network Using Genetic Algorithms

研 究 生：馬毓晴

指導教授：田伯隆　博士

中 華 民 國 102 年 6 月

數據中心網路之基於基因演算法的能源考量流量排程

Energy Aware Flow Scheduling for Data Center Network Using Genetic Algorithms

研 究 生：馬毓晴　　Student：Yu-Ching Ma

指導教授：田伯隆　博士　　Advisor：Dr. Po-Lung Tien

國 立 交 通 大 學

電 信 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Department of Communication Engineering

College of Electrical and Computer Engineer

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Communication Engineering

June 2013

Hsinchu, Taiwan, Republic of China

中華民國 102 年 6 月

數據中心網路之基於基因演算法的能源考量流量排程

學生：馬毓晴　　指導教授：田伯隆

國立交通大學電信工程研究所碩士班

摘　　要

雲端運算（cloud computing）是近年來蓬勃發展的技術之一，而作為支撐整個雲端架構的核心，數據中心（data center）的建置也十分受重視。數據中心必須支援大量的運算以及資料的儲存與傳輸，其中順暢的網路是必不可少的，而要如何選擇網路的路徑使得我們可以達到高吞吐量（throughput）及低延遲等目標並非一個簡單的問題。同時數據中心為了保證服務，必須耗費龐大的能源，導致了高昂的操作及維護成本，並產生對環境有傷害的碳排放量。所以除了維持原本數據中心該有的效能之外，還必須考慮如何減少數據中心的耗電量。

本論文中針對近年來耗電量大量成長的數據中心的網路部分來做改善。由於交換器（Switch）的開關與否是影響網路部分耗電量的主要原因，在不需要的時候關閉交換器可以有效的降低耗電量，且通過數據中心的流量是不斷變動的，所以我們可以透過良好的路由控制來改善網路的耗電量。

在短時間內快速決定路由是一個很複雜的問題，我們在本論文中選擇利用基因演算法（Genetic Algorithm）來達到此目標，基因演算法是模仿物競天擇的法則來求解最佳化問題，屬於 heuristic algorithm 的一種，具有快速的特性，代價是無法保證正確性跟精確度。而本論文中也對基因演算法做出針對欲解問題的改良，以期提高求得之解的正確性。

Energy Aware Flow Scheduling for Data Center Network Using Genetic Algorithms

Student: Yu-Ching Ma Advisor: Dr. Po-Lung Tien

Department of Communication Engineering
National Chiao Tung University

ABSTRACT

Cloud computing is one of the growing technology in recent years. Data center as the core of the supporting of the entire cloud services, the topic of how to build a data center is very important. Data center should support a large number of computing and the storage and transmission of data, which needs unblocked network. However, it's not a simple question to choose a path of the Internet which can achieve targets such as high throughput and low delay. At the same time, data centers consume huge amounts of energy to ensure performance, which causes high operational costs, and huge carbon footprints are unfriendly to the environment. Therefore, we have to consider how to reduce energy consumption and keep high performance.

This thesis focus on network equipments in the data center which have rapidly growth of energy consumption recent years. The switches contribute the largest propotion of energy consumption of network equipments, so turn off unneeded switches reduce energy consumption effectively. We can develop good routing algorithm to improve energy consumption of network equipments.

It's a complicated problem to decide routing path in a short period, so we choose genetic Genetic Algorithm to achieve our goals. Genetic algorithm is one of a heuristic algorithm. It solves the optimization problem quickly by imitating the way of the natural selection. We use fat-tree topology in our simulation, and make some improvements of GA in order to fit our problem and raise the correctness of its solution.

誌謝

本篇論文的完成，首要感謝指導教授田伯隆老師，在這兩年中，從無到有訓練發現問題、解決問題的能力，在遇到瓶頸之際不時的討論並指點我正確的方向，同時對於研究所需的資源也不吝於支持與付出。感謝柯柏宇、徐子凱學長、王櫻瑾學姐不厭其煩的指出我研究中的缺失，且總能在我迷惘時為我解惑，同學陳星豪、蕭佑霖以及學弟譚邵渝、李宗唐平時頻繁的研究討論以及有趣的生活交流，讓整個實驗室的研究氣氛非常融洽，使我能隨時保持良好的狀態進行研究。此外，感謝許多朋友如林可涓、劉姿伶、施妤青、王鈺婷、劉已加等在我疲憊沮喪的時候為我加油打氣。最後也最重要的，感謝父母一路的支持，不論是生活的幫助或是心靈方面的鼓勵，讓我能夠無後顧之憂地取得人生中重要的學識資產。

目錄

摘　　要 i

ABSTRACT ii

誌謝 iii

目錄 iv

圖目錄 vi

表目錄 viii

1　序論 1

1.1 前言 1

1.2 研究動機及目的 2

1.3 章節介紹 2

2　研究背景 4

2.1 Data Center Network 4

2.2 Fat-tree topology 5

2.3 基因演算法 6

3　研究方法 10

3.1 數學模型 10

3.2 基因演算法設計 11

3.2.1 染色體編碼 11

3.2.2 初始化 12

3.2.3 適應值函數 13

3.2.4 基因分數 13

3.2.5 選擇母代個體方法 14

3.2.6 交配方法 15

3.2.7 突變方法 16

3.2 使用 LINGO 驗證 16

4 實驗結果 17
4.1 環境與參數設定 17
4.2 使用簡單迴圈來決定要丟棄的資料 17
4.3 使用 GA 來決定要丟棄的資料 22
4.3.1 weight 的決定 22
4.3.2 模擬結果 24
4.4 基本交配方法與改良交配方法的比較 26
4.5 交配長度的影響 28
4.6 批次傳送 30
4.6.1 設定 30
4.6.2 封包大小的影響 32
4.6.3 不同 load 32
5 結論 37
5.1 研究結果 37
5.2 未來展望 37
6 參考資料 38

圖目錄

圖 1 電能的變化圖[5] 4
圖 2 Fat-tree topology 5
圖 3 基因演算法流程圖 7
圖 4 輪盤式選擇法示意圖 8
圖 5 交配策略示意圖 8
圖 6 突變策略示意圖 9
圖 7 Fat-tree 路徑示意圖 12
圖 8 染色體的編碼 12
圖 9 交換器分組示意圖 14
圖 10 基因分數計算法 15
圖 11 新子代產生方法示意圖 16
圖 12 平均吞吐量趨勢圖 18
圖 13 平均使用能量趨勢圖 19
圖 14 吞吐量標準差趨勢圖 20
圖 15 使用能量標準差趨勢圖 21
圖 16 weight 分別為 10、100、1000 的趨勢圖 23
圖 17 overflow 及 drop 分別的趨勢圖 24
圖 18 平均 f_t 的趨勢圖 25
圖 19 平均 f_e 的趨勢圖 26
圖 20 New crossover 與 2-cut crossover 比較，吞吐量 27
圖 21 New crossover 與 2-cut crossover 比較，使用能量 28
圖 22 Different crossover length compare 30
圖 23 Batch 流程圖 31
圖 24 Average flow size=4.6 33

圖 25 Average flow size=4 34
圖 26 Average flow size=3.4 35

表目錄

表 1 Power(W) of Device B in [6] 4
表 2 參數說明 10
表 3 變數說明 10
表 4 環境與參數設定 17
表 5 traffic pattern 17
表 6 traffic pattern 21
表 7 GA 與 LINGO 結果比較 21
表 8 模擬 10 次比較表 24
表 9 改良 crossover 及 2-cut crossover 的比較 26
表 10 各種長度下不同解的出現機率 29
表 11 同 load 不同 P_{gen} 及 P_{size} 的比較 32

1 序論

1.1 前言

雲端運算（cloud computing）在過去幾年間快速發展，並成為當今最熱門的科技之一。所謂的雲端運算即是基於網際網路的運算，概念來自於 1983 年 Sun Microsystems 提出的「網路是電腦（The Network is the computer）」，服務業者提供設備如主機、儲存空間等，而用戶則在任何地方都能透過網路來使用服務或存取資料，並將需要用到的計算工作都交給業者端的設備運作。對用戶來說，可減少購置設備的成本，也不需具備維護基礎設備的專業知識，對業者來說，由於程式是集中管理，可以快速的一次更新，在調整各用戶所需求的不同流量時也更加有彈性。

在雲端運算這個技術中，數據中心（data center）扮演著核心的角色。許多知名的國際企業如 Google、Amazon、Microsoft 等也都在許多地方建立及發展數據中心以便提供自己的雲端運算服務。由於所有用戶的運算需求都是交給提供雲端運算的服務業者，要怎麼快速的傳遞這些大量的資料成為十分重要的課題。除了大量的資料，要面對的難題還有必須處理各種不同的服務類型，有的服務如傳送瀏覽網頁的封包只需使用很短的時間，有些服務如模擬則需要較長的時間。為了可以提供使用者這些不同的服務需求，除了良好的網路拓樸及路由機制的建置之外，還必須使用大量的機器設備，並且依照不同類型的服務分配資源，以確保他們之間不會互相干擾來確保服務的品質。

然而數據中心早期的發展都注重在提高性能，使得有關於能源的使用量的議題一直以來都被忽略，時至今日數據中心的能源使用量已經十分龐大。大量的能源需求不僅造成成本急速上升，同時也對環境帶來巨大影響。成本增加會使的數據中心的維護變得困難，並減少獲得的利益。另外在環保意識抬頭的今天，數據中心大量的能源使用會增加碳足跡，甚至世界各國政府也都受到必須減少碳排放

量的壓力。因此我們應該開始將原本只專注在提高性能上轉換成在保證服務品質的同時又能做到更有效率的使用能源的方法。

1.2 研究動機及目的

承上小節，有些研究在針對減少能源消耗這部分，是專注於改善主機及冷卻設備的耗電量。包括如使用較低耗能的 CPU[1]、更加有效率的零件，或是在軟體方面加以改良，如 Smart cooling[2]等。

除去以上在伺服器及冷卻系統等部分所使用的能源，網路也是一個值得著手的方向。通常網路的能源消耗占整個數據中心的 10-20%[3]，而美國在 2006 年內，數據中心內僅網路部分就使用了 30 億千瓦，且從 2000 到 2006 間，網路部分的用電量上升了 14%[4]，是十分可觀的數字。所以在本研究中，我們專注在改善網路這方面的能源消耗量，我們希望能找出一種方法可以在確保數據中心內傳輸資料的暢通及穩定的同時，也能降低數據中心網路的能源消耗。由於此問題屬於 NP-hard 的問題，所以我們選擇使用基因演算法作為基礎，希望能快速可靠的來找出逼近此問題最佳解的近似解。

1.3 章節介紹

第一章 序論：

雲端運算及數據中心的演進及研究動機。

第二章 研究背景：

詳細介紹研究中有關 data center 的相關資訊及所使用的 Fat-tree 拓樸結構及基因演算法的背景知識。

第三章 研究方法：

介紹針對本研究的問題的數學模型，以及經過調整的演算法做法，並且介紹如何使用 LINGO 這個軟體來驗證。

第四章 實驗結果：

模擬的結果及各種不同方法的比較。

第五章 結論：

綜合實驗結果與討論及未來研究的方向。

2 研究背景

2.1 Data Center Network

數據中心雖然可以提供高峰的流量傳輸，但在大多數的時間裡，實際上會使用到的流量是低於最高值的，而其流量也會隨著時間（每日、每週、每月、每年）不斷的在變動。

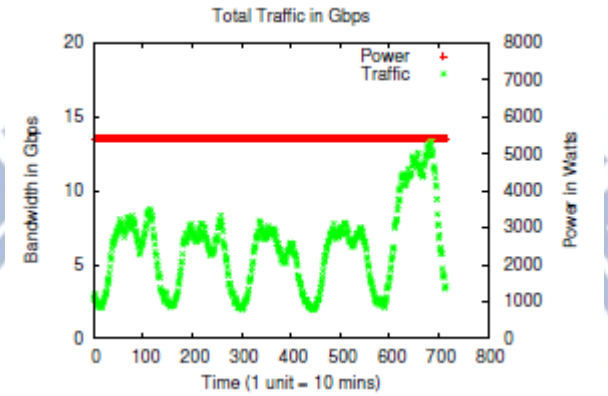


圖 1 電能的變化圖[5]

圖 1 為 292 個網站伺服器在 5 天內的流量及使用電能的變化圖[5]，可以看出流量有明顯的改變，但是使用的電能卻幾乎沒有變化（圖中的實線）。可以得知流量的大小對電能的影響並非正相關的。而在另一份能量量測的研究[6]中則比較了幾種不同流量及狀況下的交換器的用電量。根據該篇研究 Device B 的結果，可以做成表 1。

表 1 Power(W) of Device B in [6]

Active Ports	0	all	all
Traffic	0	0	1 Gbps
Power(W) of Device B in [6]	151	186	190

研究中由於篇幅不足的關係，並沒有其他 device 的詳細數據，但趨勢是相同的。因此就上表來說，我們可以看出當完全沒有流量，port 也都關閉的時候，所耗費的能量就已經快要達到流量全滿的狀況的 80%了，並且當 port 都有運作

的時候，流量從零到全滿更是只增加不到 5%。所以我們可以確認開啟交換器本身就是最消耗能量的事。所以為了達到能量最小化的目標，我們應盡量把流量集中在同一個交換器上，並將其他沒有使用的交換器關閉。實驗的設計上也會朝著這個方向來思考。

2.2 Fat-tree topology

Fat-tree 是基於 complete binary tree 發展出來的一種拓樸結構。他可以在擴大拓樸規模時，同時合理地增加網路容量，並保證頻寬可以得到最有效率的利用[7]。

一個 k-ary 的 fat-tree 有 k 個 pod，而每個 pod 裡又包含了兩層，每層有 k/2 個交換器，每個交換器有 k 個 port，較下層的交換器每個都連接到 k/2 個 server，剩下的 k/2 則是連接到 k/2 個 aggregation 層的交換器。核心（Core） 交換器有 $(k/2)^2$ 個，每個也都是 k-port，各自連到 k 個 pod。每個核心交換器的第 i 個 port 與 pod i 相連，並使得每個 pod 裡 aggregation 層連續的交換器會以 k/2 的間隔連到核心交換器上。[8]圖 2 則是 4-ary fat-tree 的範例。

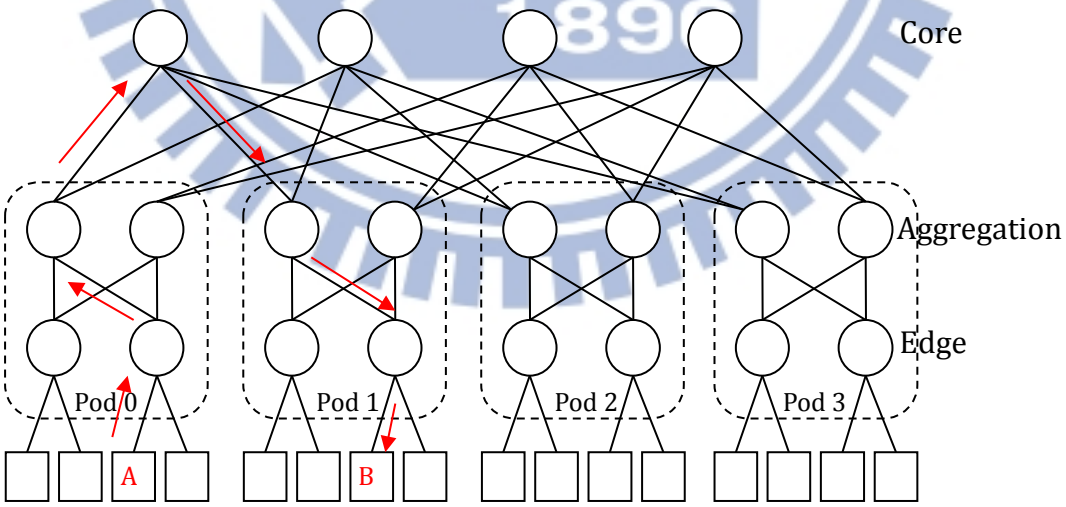


圖 2 Fat-tree topology

在傳送路徑方面，若由 A server 要傳送資料到 B server，是先往上傳，經由

Edge 層、Aggregation 層，再到 core 交換器層轉接到 B server 所在的 pod。如圖 2 的箭頭方向。其 routing 方法本來有自己的一套方法，但在本實驗中由於要使用較接近 centralized control 的方法，因此不使用原本的 routing 規則。

2.3 基因演算法

基因演算法是以達爾文的進化論為概念來執行的一種尋找最佳解的演算法，由美國密西根大學的 John Holland[9]所提出，其中心思想為「物競天擇，適者生存」，在許多個體之中，依照每個個體對環境的適應性來篩選，保留下較能適應環境的一群個體，也就是在該環境較有競爭力的個體，並且隨著一代一代的演化重複這樣的篩選過程，最終可找到最適合在該環境存留的個體。

對照到演算法上，將欲解的最佳化問題的可能解組合稱為染色體（chromosome），而基因（gene）則是組成染色體的基本單位，每個個體（individual）的染色體都不同，並依照最佳化問題設計可用來評估該個體適應程度優劣的適應值（fitness），許多的個體形成一個族群（population），接著在族群內經過挑選（selection）、交配（crossover）、突變（mutation）等運算，產生出下一個世代（generation）。隨著每個世代的演化，每新的一代的染色體適應性會漸漸變得更加優秀，也代表更接近我們所需要的最佳解。

在開始基因演算法之前，必須先決定 1. 染色體的編碼方式，2. 適應函數（fitness function）。

染色體的編碼分成三種[10]，實數編碼、二進制編碼、符號編碼。實數編碼為直接使用時數填入代表染色體的陣列中。二進制編碼是將數字轉換為 0 與 1，但這麼做會造成染色體的長度增加，衍生出可能不易收斂的問題，所以比較常使用在參數較少的情況下。符號編碼則是用符號（如英文字母）代替我們的目標。

適應函數是用來評估每個個體適應環境的程度，在演算法中扮演著演化論裡「天擇」的角色，因此必須好好設計適應函數讓染色體可以依照我們所期望的方向演化。

圖 3 為基本的基因演算法流程圖：

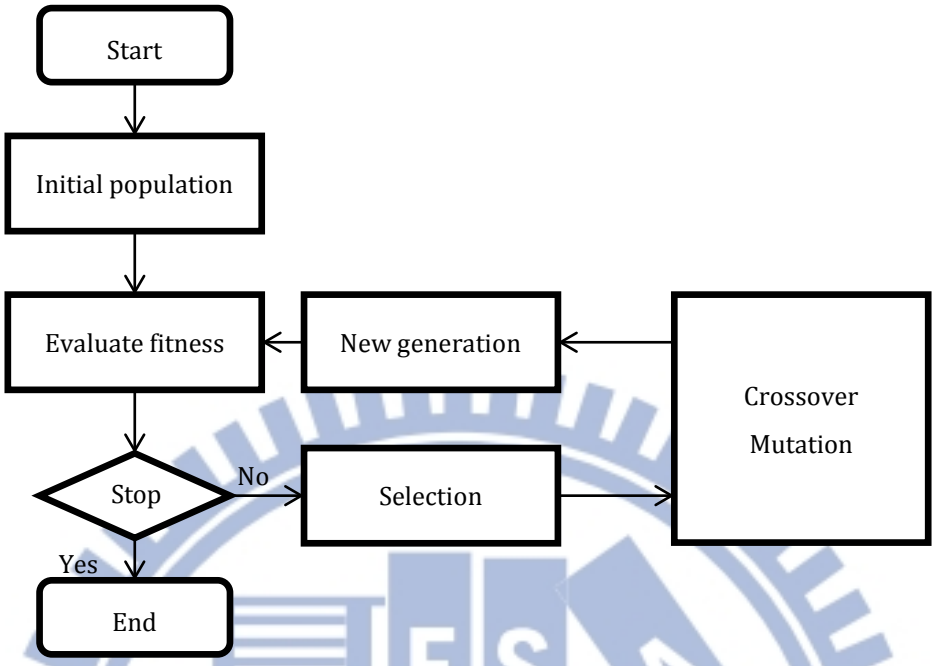


圖 3 基因演算法流程圖

- Initial population：初始化產生第 0 代族群，產生的方法通常為在可能的解空間中亂數產生，另外也可以依照原始的資料分布給值，有可能可以加速程式收斂，但也可能使演算法找到的解落在局部最佳解（local optimum）而非全局最佳解（global optimum）。
- Evalute fitness：評估每個個體的適應值，以作為接下來挑選個體的依據。適應值表現越佳的個體越容易被選中演化。計算完適應值後，判斷整個程式是否達到終止條件，終止條件一般是程式已收斂或已達到最初設定的要求，若是則終止，若否則進行選擇（selection）。
- Selection：選擇將會被保留並做運算的個體，通常有兩種選擇法：輪盤式選擇[11]及競爭式選擇[12]，輪盤式選擇的概念是將個體的適應值依照優劣比例畫成圖，如圖 4，接著再模擬以飛鏢射輪盤的方式來挑選，因此適應值越佳在圖上占的面積就越大，也就越容易被選中。競爭式選擇則是先亂數挑選幾個個體，再從其中選出適應值較佳的個體。

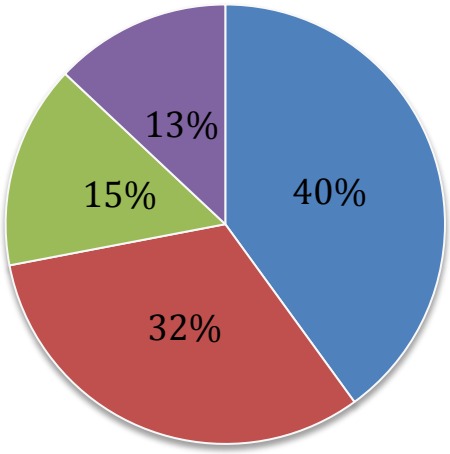


圖 4 輪盤式選擇法示意圖

- Crossover：將兩個通過選擇機制的個體的染色體各取一段，並合成為新的個體，此運算稱為交配。利用交配來產生新的一代，也稱子代，用來產生的個體則稱為母代。此運算是期望透過融合兩個個體的基因來產生更具競爭力的子代，當然也有變的更差的可能性，但這樣的個體將會在下一個迴圈被剔除掉。一般常用的有單點[13]、雙點[14]及多點交配[15]等。見圖 5。

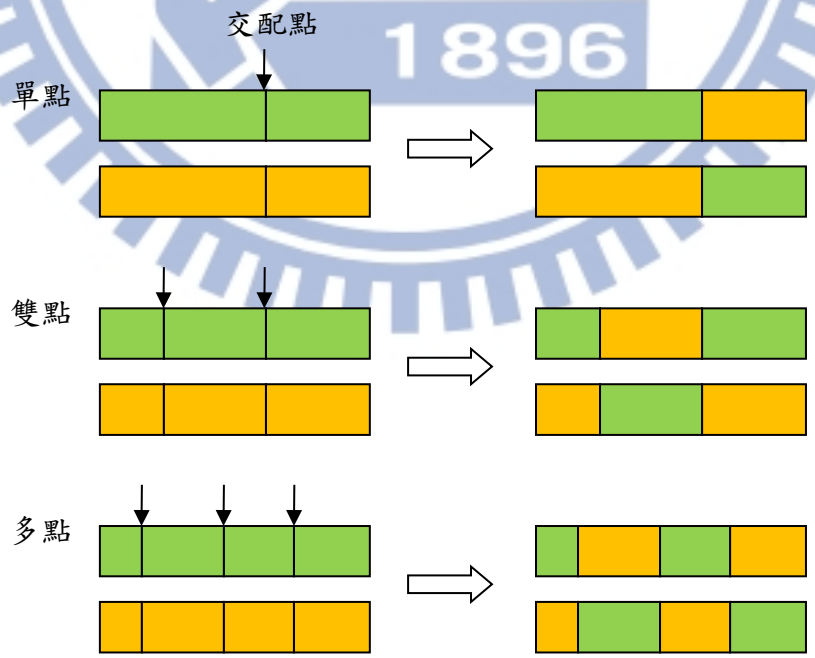


圖 5 交配策略示意圖

- Mutation：突變運算對象是 crossover 之後產生的子代個體，依照突變機率（通常機率很低)使得個體內的基因發生變化，如圖 6。如果是二進制編碼，則 0 變 1，1 變 0；如果是實數編碼，則在解的範圍內亂數產生。突變的目的是擴大解空間的搜尋範圍，並避免落入局部最佳解的狀況。但跟 crossover 一樣不能保證經過此步驟的個體會變個更好或更差。

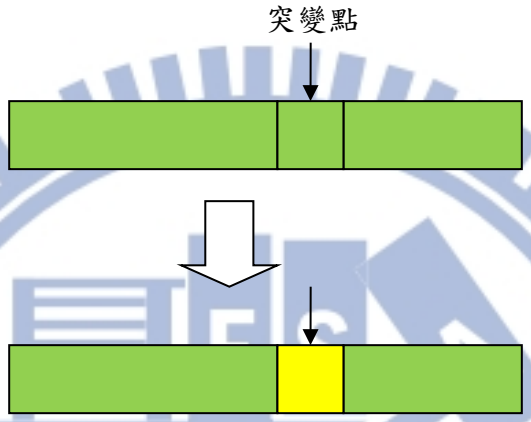


圖 6 突變策略示意圖

3 研究方法

3.1 數學模型

表 2 參數說明

參數	
符號	意義
N	所有交換器的集合
E	所有鏈結的集合
R	所有 request 的集合
P_r	Request r 從傳送端到接收端可使用的所有路徑的集合
E_n^{in}	所有進入交換器 n 的鏈結的集合，$E_n^{in} \subseteq E$
t_r	Request r 的封包大小
c_e	鏈結 e 的容量
α_n	當交換器 n 開啟時所使用的能量
β_n	當交換器 n 上每通過 1 flow 所使用的能量
K	penalty

表 3 變數說明

變數	
符號	意義
$x_{r,p}$	當 request r 選擇路徑 p 時為 1，反之為 0
f_e	通過鏈結 e 的總流量
d_r	當 request r 可被送出時為 1，反之為 0
γ_n	紀錄交換器 n 所使用的總能量
u_n	紀錄交換器 n 上通過的總流量
$\delta_{r,p,e}$	當 request r 使用路徑 p 通過鏈結 e 時為 1，反之為 0

$$Minimize: \sum_{n\in \mathrm{N}} \gamma_n + K\left(\sum_{r\in R} t_r - \sum_{r\in R} t_r \cdot d_r\right) \quad \text{.....(1)}$$

$Subject\ to$:

$$\sum_{p\in P_r} x_{r,p} = 1 \ \ \forall r \in R \quad \text{.....(2)}$$

$$\sum_{r\in R, p\in P_r} \delta_{r,p,e} \cdot x_{r,p} \cdot t_r \cdot d_r = f_e \ \ \forall e \in E \quad \text{.....(3)}$$

$$0 \le f_e \le c_e \ \ \forall e \in E \quad \text{.....(4)}$$

$$u_n = \sum_{e\in E_n^{in}} f_e \ \ \forall n \in N \quad \text{.....(5)}$$

$$\gamma_n = \begin{cases} \alpha_n + \beta_n \cdot u_n & , if\ u_n > 0 \\ 0 & , if\ u_n = 0 \end{cases} \quad \forall n \in N \quad \text{.....(6)}$$

$$x_{r,p} \in \{0,1\} \ \ \forall r \in R, \forall p \in P_r \quad \text{.....(7)}$$

$$d_r \in \{0,1\} \ \ \forall r \in R \quad \text{.....(8)}$$

式(1)為目標函數，前一項為網路中所有交換器能量使用的總和，後一項為沒被送出的 request 總流量，並乘上一個很大的值 K 作為 penalty。式(2)為限制每個 request 至少且只能選擇 P_r 中的一條路徑。式(3)計算通過鏈結 e 中的總流量。式(4) 限制鏈結 e 中的總流量在 0 到鏈結的最大容量之間。式(5)計算通過交換器 n 的總流量。式(6)計算交換器 n 的總能量。式(7)及式(8)則是限制$x_{r,p}$及d_r為 binary。

3.2 基因演算法設計

由於直接使用基本的基因演算法不夠切合我們的需求，以下介紹如何針對我們的問題來設計基因演算法中的各種機制。

3.2.1 染色體編碼

在此問題中，我們利用染色體來記錄及選擇每個 request 會經過的路徑。當

染色體過長的時候，會使得基因演算法在運作的時候不容易收斂。此處將 fat-tree 的架構由 Core、Aggregation、Edge 對應成 Layer 1~3 來討論，觀察 fat-tree 的架構，可以發現當一個 request 的 source 及 destination 是確定的情況下，我們只需知道 Layer 3 使用的是哪一個交換器，即可知道整條路徑是如何傳送的。如圖 7。

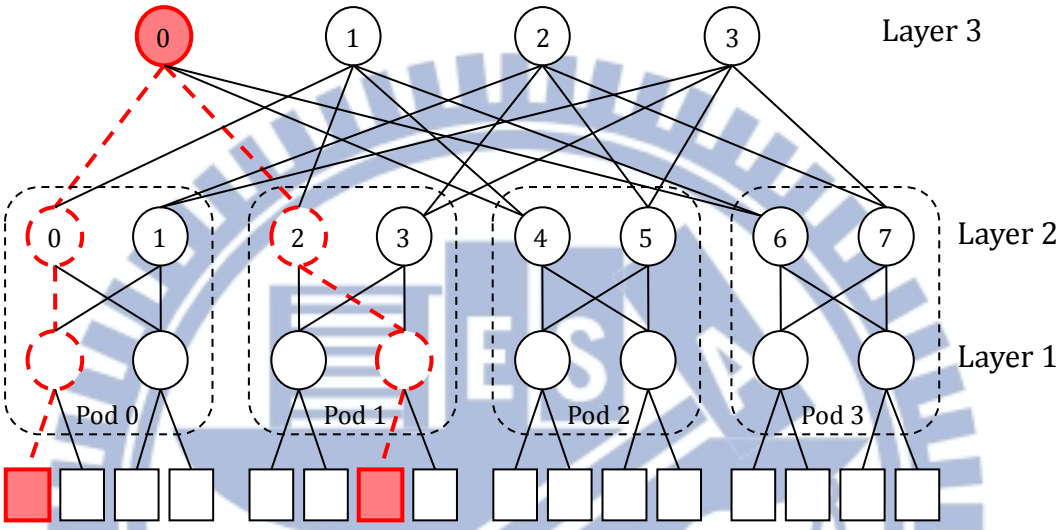


圖 7 Fat-tree 路徑示意圖

因此我們將交換器編號，每個染色體的長度設定成跟欲傳送的的 request 數量一樣。從第一個 server 的 request 開始依序排列，紀錄每個 request 所選擇的交換器編號，如圖 8 所示。

n：request 的數量

I_k：第 k 個 request 所經過 Layer 3 的 Switch 編號

圖 8 染色體的編碼

3.2.2 初始化

由於我們的染色體編碼是使用 Layer3 交換器的編號，且 fat-tree 又具有不論選擇哪個交換器都可以互通的特性，因此在一開始產生第 0 代的族群時，就亂

數產生可能的交換器編號填入染色體內。

3.2.3 適應值函數

我們的目標是要找出最節省電力的 routing 路徑，因此將行經路徑所使用的能量百分比設為適應值 f_e，然而為了避免演化到最後變成所有 request 都不傳送以達到總能量最低（即全部交換器皆關閉）的情況，我們多設了一個適應值 f_t 來計算吞吐量，並且 f_t 的優先順序高於 f_e，以確保最後結果的解會先滿足吞吐量再進一步將能量最小化。

$$f_t = \text{實際傳送的總流量} / \text{所有 request 的總流量} (\%)$$

$$f_e = \text{實際使用能量} / \text{能量最大值} (\%)$$

所以比較 f_t 的時候是越大越好，而比較 f_e 的時候是越小越好。

3.2.4 基因分數

基因分數是針對染色體中的基因，在本實驗中是指交換器的編號，分別給予分數，以用來在執行基因演算法的運算時，可以迅速的判斷此交換器相對其他交換器的優劣趨勢。

前面提過我們應盡量將流量集中在同一個交換器上面，所以我們就直接使用各交換器上流過的流量作為該交換器的分數。流量越多，分數也就越高。再來我們觀察拓樸結構可以發現，交換器又可以分成不同的組別，如圖 9 所示，雙實線屬於組別 0，虛線屬於組別 1。假設至少需使用 2 個 Layer 3 的交換器，使用同一個組別的交換器會比使用不同組別的交換器還更省能量。所以我們將組別的流量也加入分數中一起考慮。而組別的流量則是計算 Layer 2 中同組別的交換器流量總和，乘上權重後加在 Layer 3 交換器的分數中。

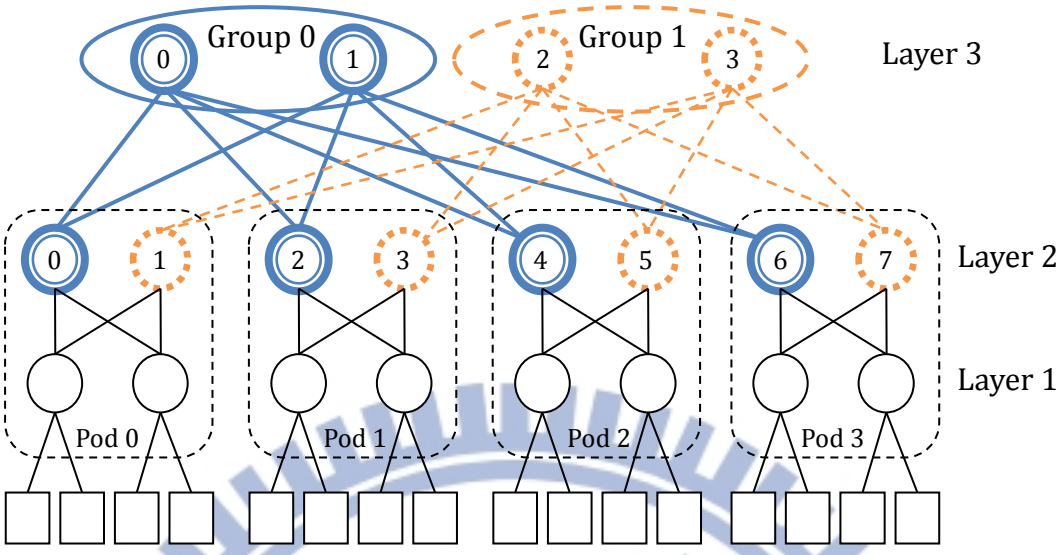


圖 9 交換器分組示意圖

$F_{k,s}$:流入 Layer k 中 交換器 s 的總流量 ex: $F_{3,0}$ 代表 Layer 3 交換器 0 的流量

G_n: Layer 2 中屬於組別 n 的總流量

$C_{k,s}$: Layer k 中交換器 s 的分數

W：weight

$$G_n = \sum F_{2,s} \quad , \text{if } n = 0, s = \{0,2,4,6\}$$

$$, \text{if } n = 1, s = \{1,3,5,7\}$$

$$C_{3,s} = F_{3,s} + W \times G_n \quad , \forall s \in n$$

需要注意的是此分數並非代表絕對的好壞，因為必須考慮吞吐量，所以流量集中反而可能造成 overflow 的狀況。

3.2.5 選擇母代個體方法

在進入交配運算前，需要選擇兩個染色體作為母代，本研究中使用較類似於前面介紹的競爭式選擇法。

一開始隨機選擇兩個染色體，先比較他們的 ft，若相同，再比較 fe，挑出其

中較好的一個染色體，接著重複一次上述動作，再挑出另外一個染色體，以這兩個染色體作為母代，產生出兩個新的一代的染色體。產生方法為下小節介紹的交配運算及下下小節的突變運算。

3.2.6 交配方法

以兩點交配為基礎，配合上小節設計的基因分數。首先先決定欲交配的基因長度 L，染色體長度為 n，所以一個個體會有(n-L)+1 段長度 L 的不同基因，每段的分數即為其中 L 個基因分數的總和。又，在不同的個體間，同一個交換器的分數也是不同的，所以在交配的時候，必須先選定作為主體的個體，在計算分數時使用該個體的分數來計算。

假設母代的兩個個體分別為 A 及 B ，染色體長度為 12，L 為 5，先以 A 為主，計算 A 及 B 裡 8 段的分數，如圖 10 所示。

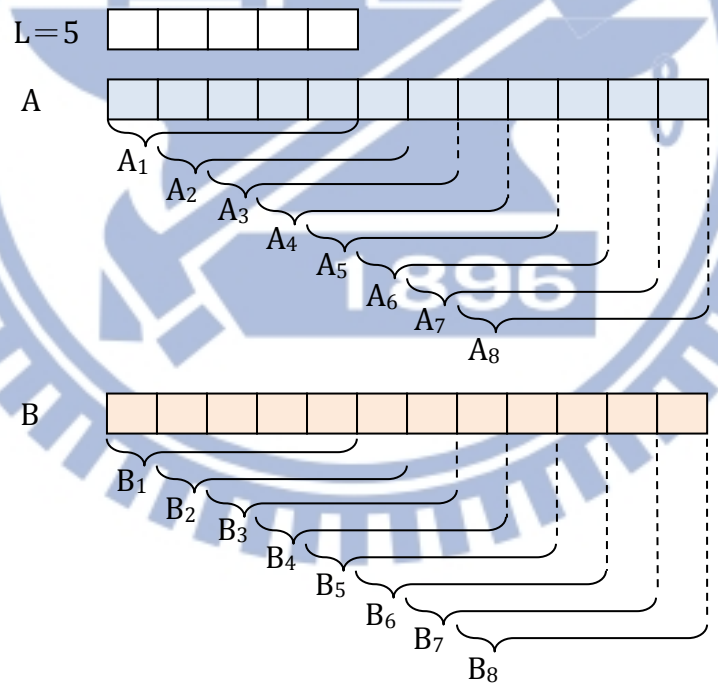


圖 10 基因分數計算法

比較 A_1─B_1、A_2─B_2...A_8─B_8，從中找出 A 較差而 B 較好，且差距最大的一段。此處先假設是 A_5─B_5 這一對，則將 B_5 取代 A_5，產生出一個新的子代 A’，如圖 11 所示。

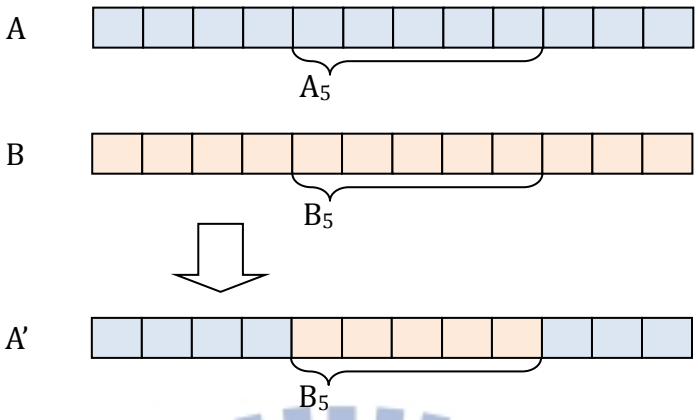


圖 11 新子代產生方法示意圖

產生出 A'後，改為以 B 為主體，重複一次計算各段基因分數及比較的動作，即可產生出 B'。產生出的 A'及 B'接著進入下小節的突變運算。

3.2.7 突變方法

設定突變機率為 0.05。並讓染色體中的每個基因都有突變的機會，而非只有其中一個基因會突變。

另外為了增加搜尋到 global optimum 的機會，我們利用族群內適應值的標準差來判斷整個族群內的個體是否已趨向一致，因為當所有個體都十分相似的情況下，藉由交配此手段想要產生新的解釋很困難的。所以當我們判斷個體間已經太過相似時，將增加突變機率以期搜尋到新的解。

3.2 使用 LINGO 驗證

LINGO 是一套專門用於求解最佳化問題的軟體，可以用來求線性、非線性(convex and nonconvex)、二次、二次限制和整數最佳化的解。LINGO 有自己獨特的語言，可以表達出各種問題的公式，並且容易閱讀及修改。在輸入資料時，也可以經由檔案讀取，並將求解的結果輸出。於是我們利用此套軟體，將上述的數學模型輸入以驗證我們使用的演算法的正確性，但只能驗證到 4-ary 的 fat-tree 結構，若將網路的 size 再放大，參數就會超出軟體限制，執行所需花費的時間也無法估計。

4 實驗結果

4.1 環境與參數設定

表 4 環境與參數設定

Number of servers	16
Number of layer 1 switches	8
Number of layer 2 switches	8
Number of layer 3 switches	4
Link capacity	5

我們使用 4-ary 的 fat-tree 拓樸來做模擬，環境的各項設定如表 4。傳送 packet 的目的 server 與 packet 的大小皆為亂數決定，封包大小∈{0,1,2,3,4,5}，每一個 server 最多只會產生一個 packet，意即可能會有 2 個以上的 packet 由同一個 server 接收，但不會有 2 個以上的 packet 是由同一個 server 送出的。

4.2 使用簡單迴圈來決定要丟棄的資料

首先設置一組 request 來執行，如表 5。

表 5 traffic pattern

Source	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Destination	1	0	0	2	0	6	4	5	1	0	0	2	0	6	4	5
size	3	1	2	2	1	5	5	3	1	1	1	4	1	3	2	5

其中交換器 1, 2, 4, 9, 10, 12 皆要將 packet 傳送至交換器 0，而其封包大小總和為 7，已超過鏈結容量，因此必定會有 overflow；或者若交換器選擇的不好，也會造成 overflow。為了解決此問題，我們使用迴圈去掃描拓樸裡的每條鏈結，當發現鏈結上的流量已經超過其容量時，便從會消耗最多能量的 request 開始捨去，且不管該 request 裡的封包大小有多少都一次捨棄（以下稱捨棄的流量總數為 drop）。沒有捨棄的 packet 即為可傳送出去的 packet，於是我們可利用 drop

值來計算吞吐量。

我們設定 population size 為 50，generation 為 50，mutation probability 為 0.05 來模擬，以下為使用前述 traffic pattern 模擬 10 次的圖，並如同前面提過我們使用兩段的 fitness 來控制演化方向，我們將每個 generation 的平均吞吐量及平均使用能量畫成圖來比較。圖 12 為平均吞吐量，圖 13 為平均使用能量。

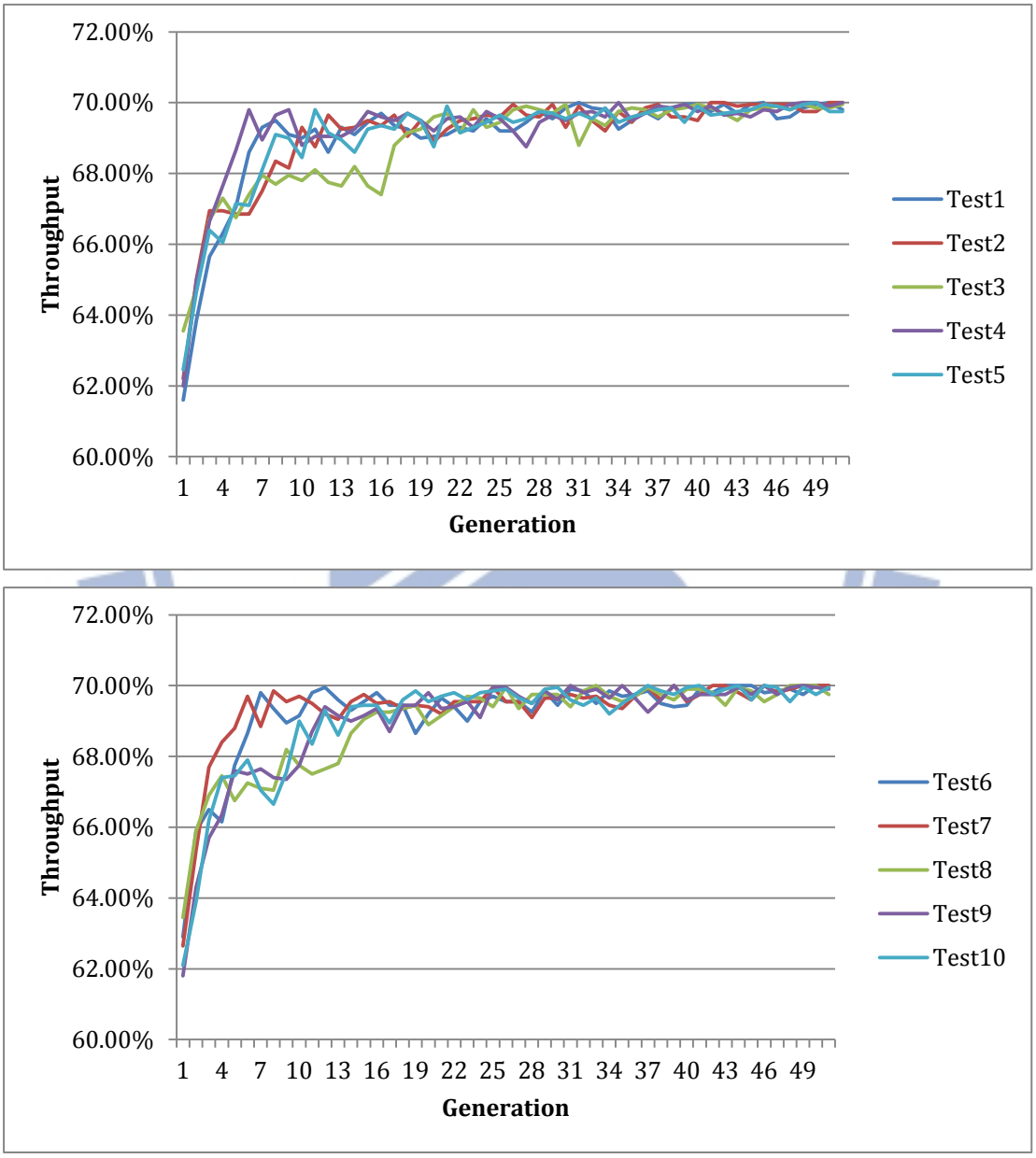
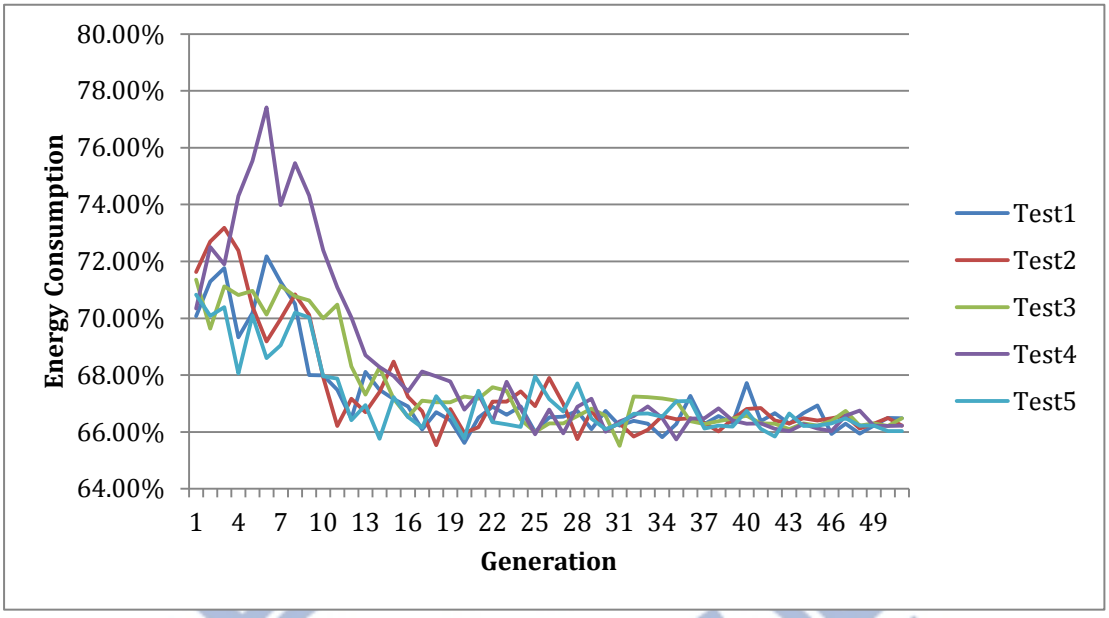


圖 12 平均吞吐量趨勢圖



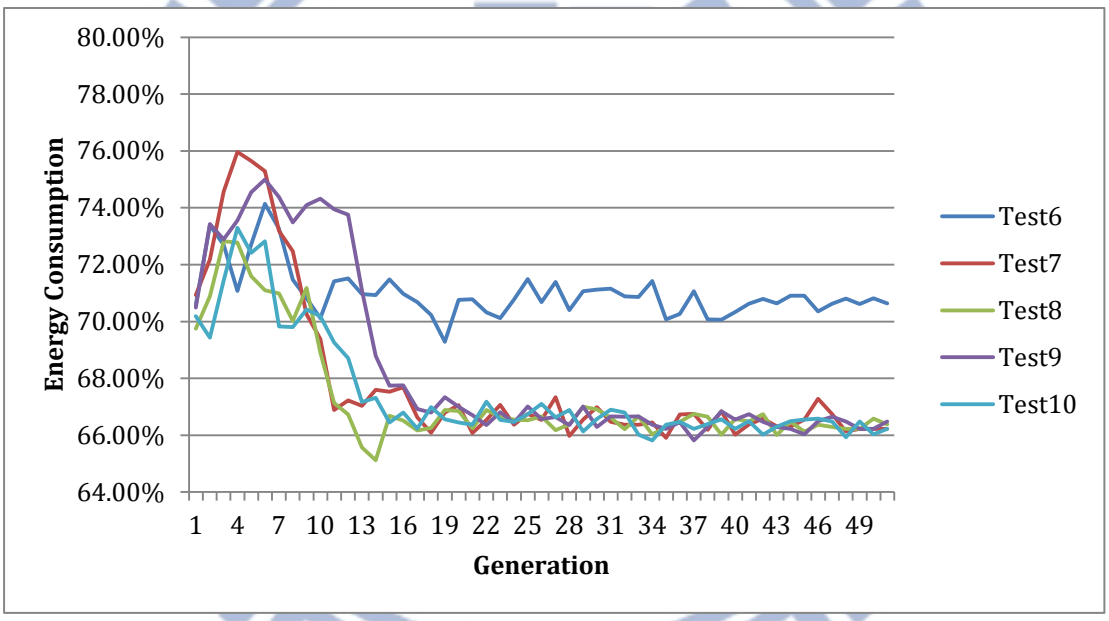
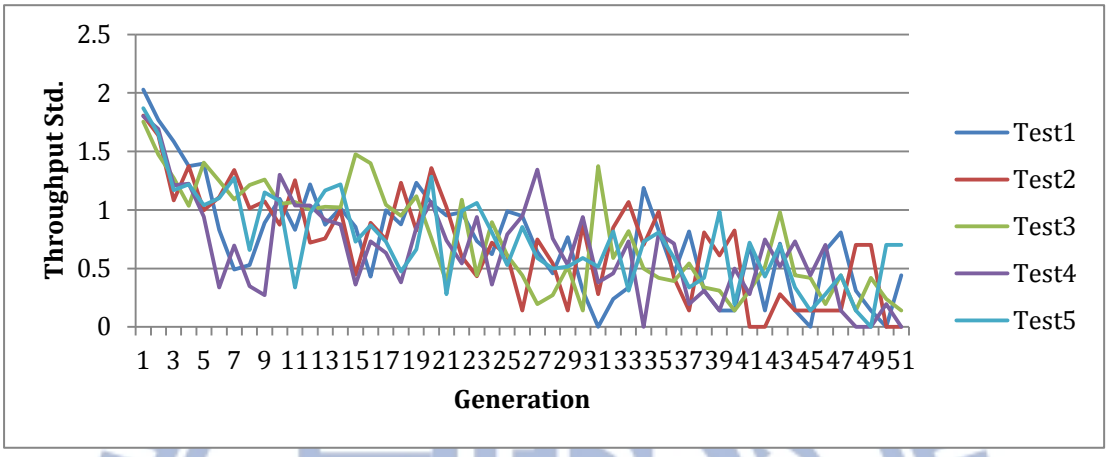


圖 13 平均使用能量趨勢圖

由圖可看出，吞吐量有隨著 generation 而增加，使用能量也逐步減少，並且都有收斂的跡象。進一步觀察吞吐量及使用能量的標準差，也可以看到兩者的標準差都有降低的趨勢（圖 14 及圖 15），代表每一代內部的個體差異有漸漸變小，也初步驗證了使用基因演算法的確能將解逐步往好的方向搜尋，而非隨機的找解。



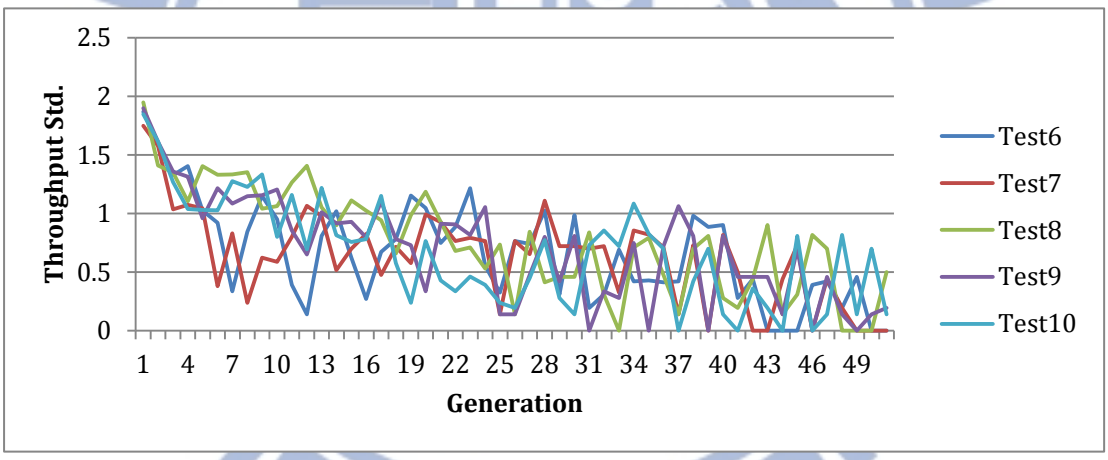
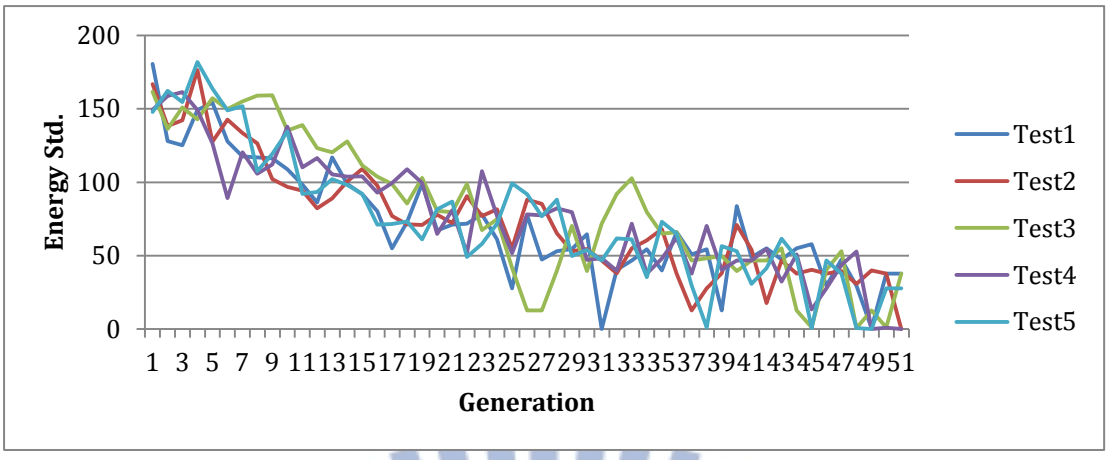


圖 14 吞吐量標準差趨勢圖



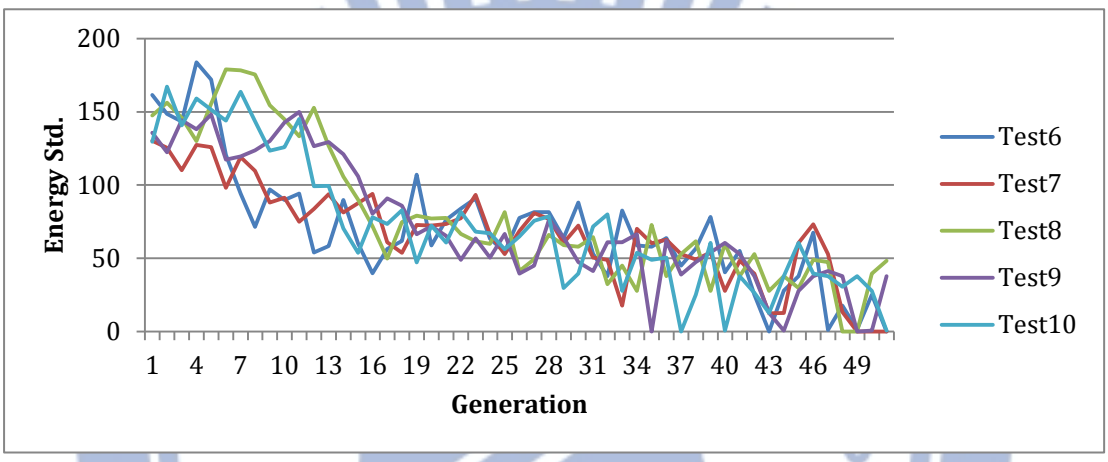


圖 15 使用能量標準差趨勢圖

然而在處理 overflow 的情況時採用迴圈的方式將超過的部分一個一個拿掉的方法，在某些狀況中所找到的並非最佳解。例如：假設有一組 traffic pattern 如下表 6。

表 6 traffic pattern

Source	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Destination	1	4	3	2	8	9	15	11	5	6	4	13	10	6	12	7
size	3	5	2	1	5	3	4	3	1	2	5	1	2	3	3	3

以此 pattern 放入 GA 及 LINGO 分別運算，可得表 7 結果。

表 7 GA 與 LINGO 結果比較

	throughput	Energy
GA	89.1304%	89.875%
LINGO	89.1304%	85.375%

可以看出 GA 的吞吐量雖然跟 LINGO 一樣，但使用能量卻較高，比較 GA 跟 LINGO 的紀錄後發現，兩者差在捨棄掉不同的 request。在該 pattern 裡，交換器 1→4 跟交換器 10→4 的 request 會產生 external blocking 的問題。當使用 GA 時，相同大小的 request 由於迴圈的關係會選擇 source 編號較大的 request 來捨棄，也就是讓交換器 10→4 不要送出。然而在此 pattern 中，限制交換器 1→4 不僅可達到一樣的吞吐量，對能量使用的效率也更高。為了解決這個問題，我們嘗試使用下小節所介紹的利用 GA 來決定要丟棄的資料。

4.3 使用 GA 來決定要丟棄的資料

我們嘗試新增一個代表「不傳送」的編號 $(k/2)^2$，此處由於我們的 k=4，所以當基因值等於 4 的時候，即代表該 request 將不被傳送。

此外，在不使用迴圈後，一組解會包含 3 個資訊：(1) drop、(2) overflow、(3) 使用能量。Drop 與 overflow 同屬於會影響吞吐量的參數，於是在此法中我們將 fitness 重設如下：

$f_t = \text{drop} + w \times \text{overflow}$　　w：weight

$f_e = {}^{\text{實際使用能量}}\!/_{\text{能量最大值}}\ (\%)$

必須注意的是，一個可行的解必須使 overflow 為 0，否則放入網路中就會有 congestion 的問題，所以我們將其乘上一個權重，目的為加速 overflow 的減少。

4.3.1 weight 的決定

為了保證新的 f_t 在 GA 裡也可以良好運作，並決定 weight 該使用哪個數級較適合，我們暫時先不考慮使用的能量，只保留 f_t 來模擬。圖 16 為 f_t 使用 3 個不同的 weight 的趨勢圖，圖 17 則為 overflow 與 drop 拆開後的趨勢比較。

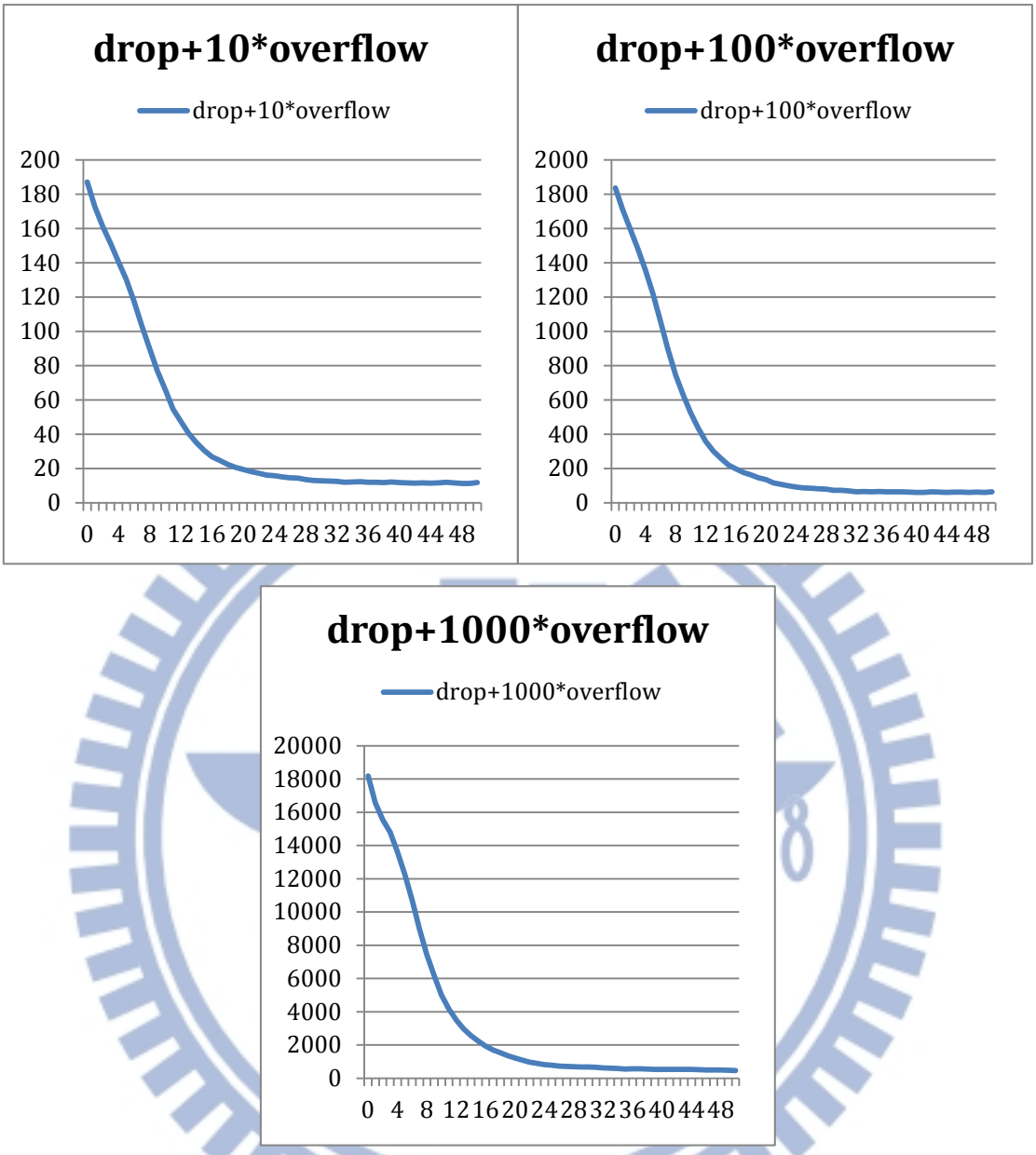


圖 16 weight 分別為 10、100、1000 的趨勢圖

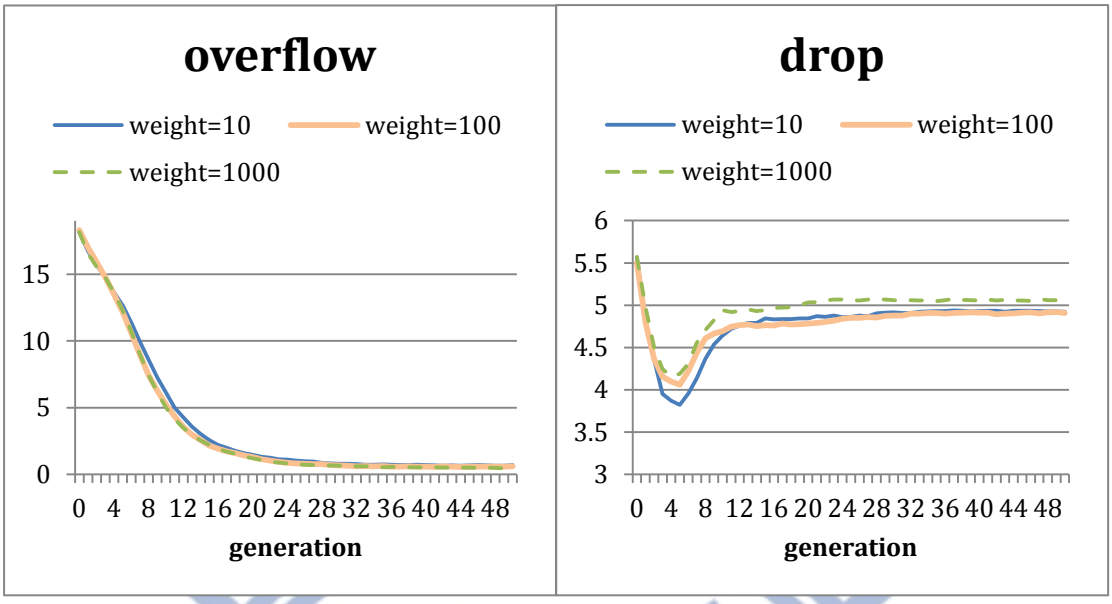


圖 17 overflow 及 drop 分別的趨勢圖

可以看出 3 種 weight 的差異不大，但為了保險起見，我們還是選擇 weight 為 1000 來做模擬。

4.3.2 模擬結果

使用上小節提過會產生問題的 pattern 來模擬（表 4），並比較上小節方法與本小節方法的差異。表 8 為兩個方法各模擬 10 次的結果。

表 8 模擬 10 次比較表

	Throughput		Energy	
	迴圈	GA	迴圈	GA
Test1	89.1304%	89.1304%	89.875%	89.875%
Test2	89.1304%	89.1304%	89.875%	89.875%
Test3	89.1304%	89.1304%	89.875%	85.375%
Test4	89.1304%	89.1304%	89.875%	89.875%
Test5	89.1304%	89.1304%	89.875%	89.875%
Test6	89.1304%	89.1304%	89.875%	89.875%
Test7	89.1304%	89.1304%	89.875%	89.875%

Test8	89.1304%	89.1304%	89.875%	89.875%
Test9	89.1304%	89.1304%	89.875%	85.375%
Test10	89.1304%	89.1304%	89.875%	89.875%

兩種方法找到的最高吞吐量是一樣的，但改用 GA 可找到 Test 3、9 這種較低的使用能量，證明此法是有效的，然而其找到最佳解的 hit rate 在此 pattern 中只有 20%，機率較低。原因也有可能是低能量解的組合本來就比較少，所以找到的機率低。但考慮到 GA 多增加了一個可選基因，每個基因的可能選擇由 4 個變成 5 個，若染色體長度為 n，則解空間由 4^n 變成 5^n，複雜度增加，精準度自然也下降。圖 18 及圖 19 為平均 f_t 及平均 f_e 的趨勢圖，因 f_t 包含 overflow 在內，故沒有轉換成吞吐量。

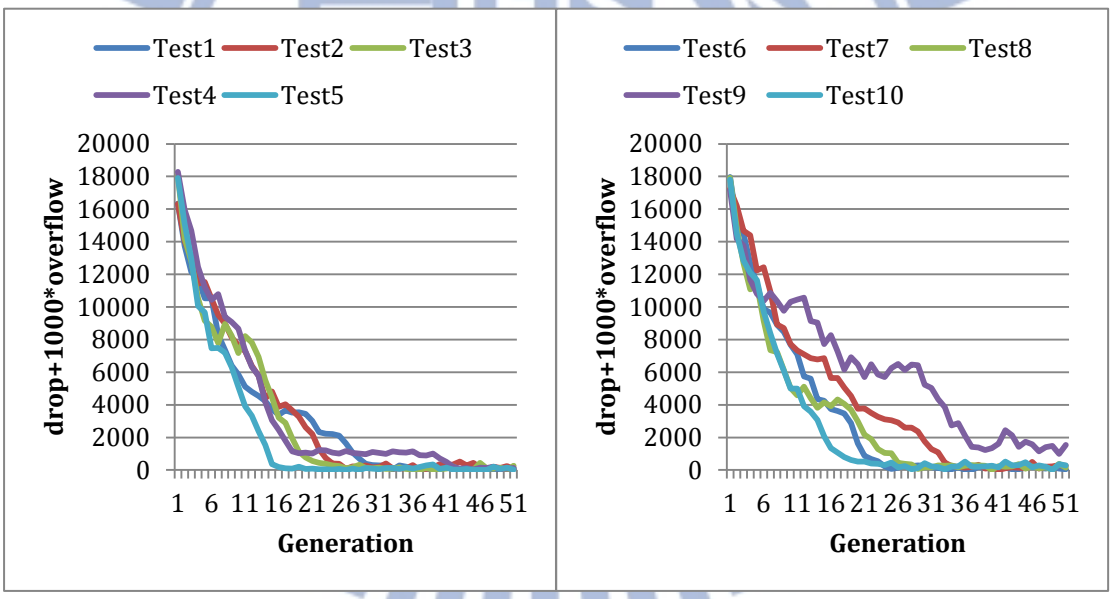


圖 18 平均 f_t 的趨勢圖

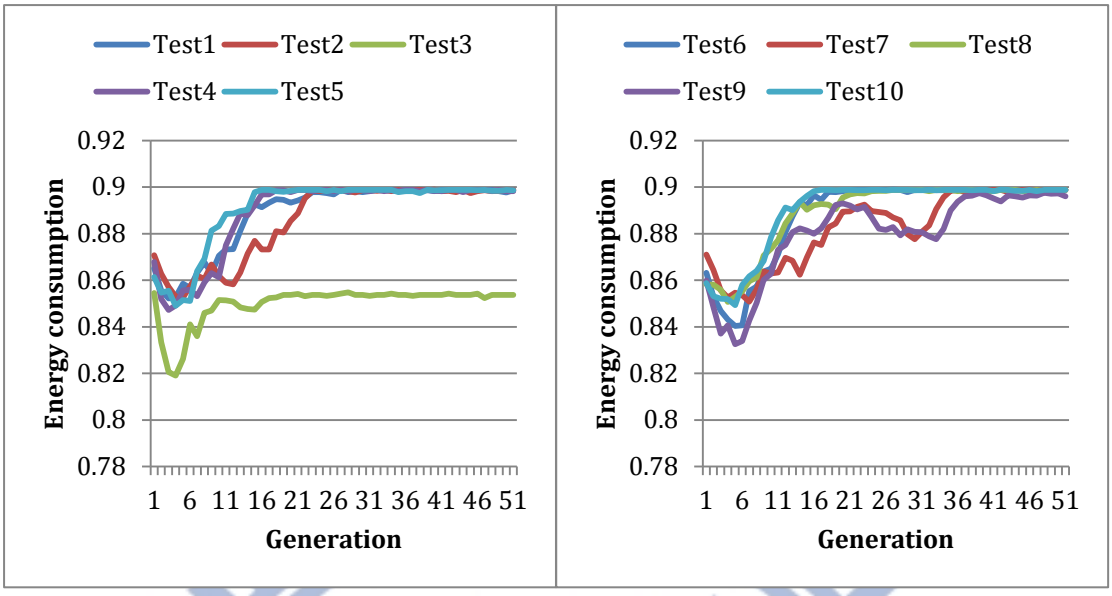


圖 19 平均 f_e 的趨勢圖

4.4 基本交配方法與改良交配方法的比較

基本的交配方法是在染色體中隨機選擇交配點並交換兩個個體的部分基因，為了使交配這個運算更具有意義，我們以 2-cut crossover 為基礎，搭配基因分數的計算，改良了交配的方法，並於前面的 3.2.6 節中做過介紹。

我們分別用 2-cut crossover 與改良 crossover 模擬同一個 pattern 100 次，把這 100 次的模擬結果依照數值統計其出現次數如表 9。

表 9 改良 crossover 及 2-cut crossover 的比較

	solution			
	Energy	Throughput	2-cut crossover	改良 crossover
好	64.81%	90.9%	39%	66%
↑	66.85%	90.9%	27%	17%
↑	68.89%	90.9%	17%	9%
↑	70.93%	90.9%	14%	7%
差	72.97%	90.9%	3%	1%

可以看出經過改良後，找到最佳解的機率由 39%提升到 66%，表示此種改

良的交配方法對增加演算法整體的效益的確有幫助。以下為 2-cut crossover 與改良 crossover 的平均吞吐量（圖 20）及平均使用能量（圖 21）的趨勢圖比較。

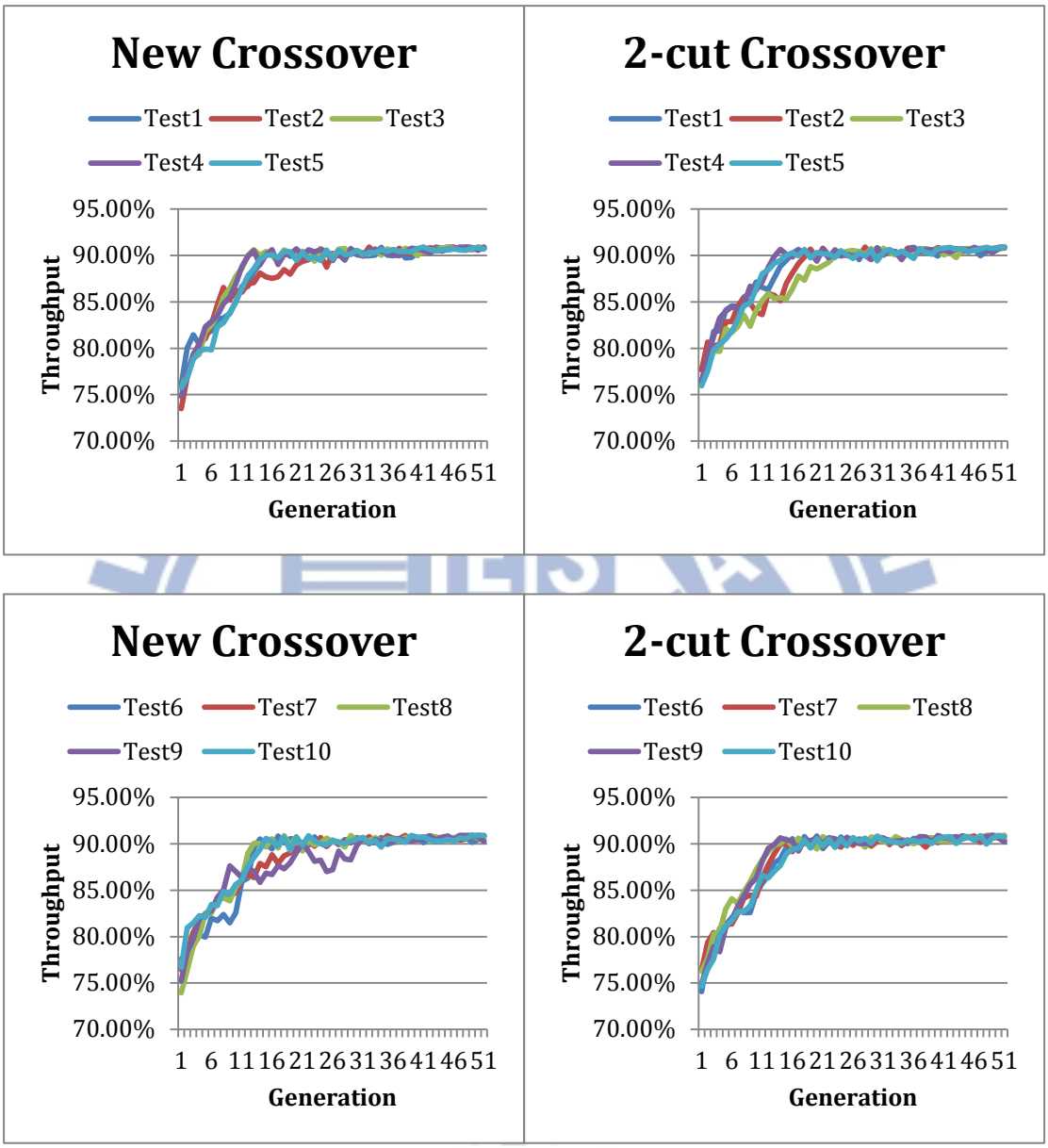


圖 20 New crossover 與 2-cut crossover 比較，吞吐量

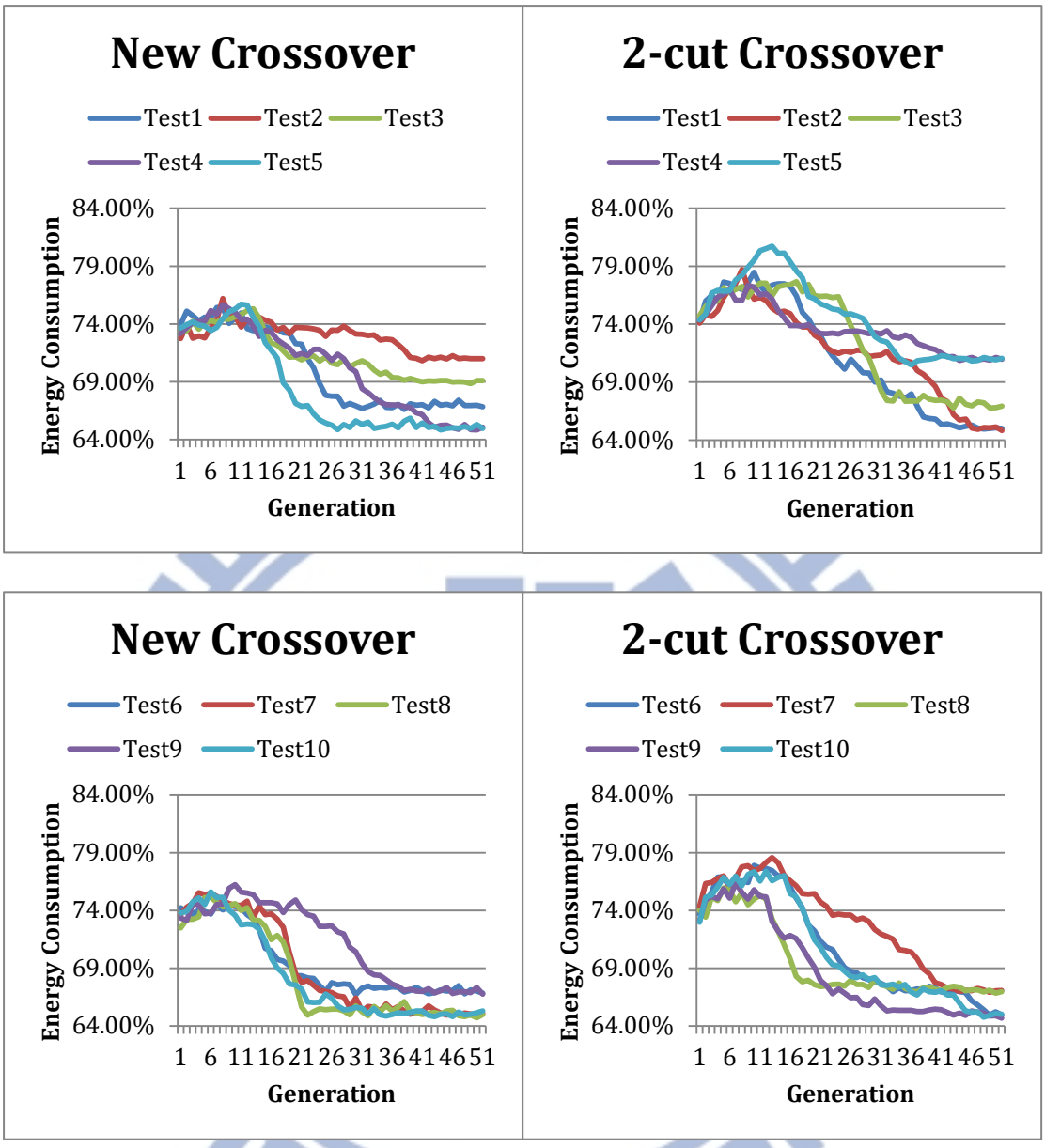


圖 21 New crossover 與 2-cut crossover 比較，使用能量

兩法在吞吐量的趨勢表現的差不多，而在使用能量方面則可看出 2-cut crossover 的整體使用能量較高一點，收斂速度也較慢。

4.5 交配長度的影響

延續上小節對改良 crossover 的討論，本小節將比較在交配時選擇不同長度對演算法的影響。因為我們改良的 crossover 的想法是將染色體中一段較不好的基因取出代換成較好的基因，而這一段不好的基因的長度該如何選擇就是本小節

嘗試的目標。

我們分別將 crossover 的長度設定成 1~15，以及在[1,15]區間用 uniform distributed 的正整數亂數，另外由於發現在長度約 4~7 之間結果較好，我們另外使用 normal distribution（mean=6）來決定長度。每種解的統計機率結果如下表 10。(此處只放幾種較好的解的機率)

表 10 各種長度下不同解的出現機率

	Throughput	90.9%	90.9%	90.9%	90.9%
	Energy	64.81%	66.85%	68.89%	70.93%
Length =1		56%	18%	9%	13%
Length =2		59%	12%	15%	11%
Length =3		62%	20%	3%	12%
Length =4		67%	11%	9%	8%
Length =5		69%	14%	8%	6%
Length =6		75%	14%	6%	5%
Length =7		72%	13%	8%	7%
Length =8		70%	10%	16%	4%
Length =9		61%	14%	15%	10%
Length =10		53%	17%	20%	10%
Length =11		47%	21%	12%	19%
Length =12		45%	19%	20%	15%
Length =13		48%	16%	18%	18%
Length =14		22%	27%	19%	28%
Length =15		25%	31%	16%	21%
random		63%	18%	11%	7%
Normal (mean=6)		74%	12%	8%	4%

取其中吞吐量為 90.9%、使用能量為 64.81%的解出現機率作圖 22。

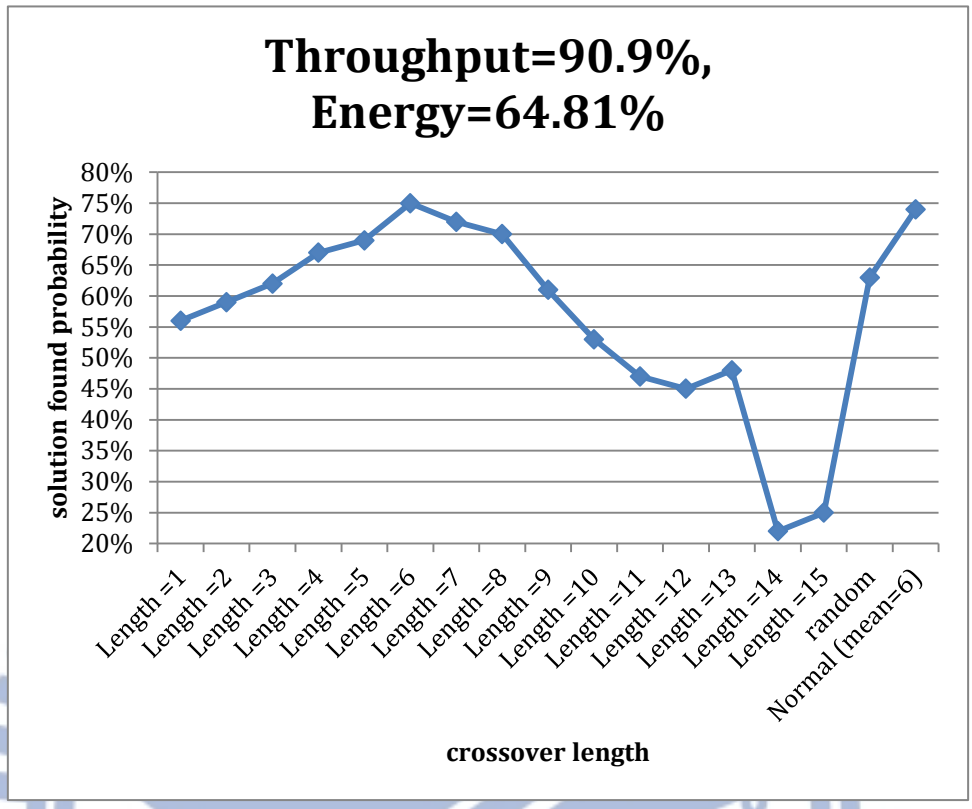


圖 22 Different crossover length compare

除了長度 4~7 之外，random 的表現尚可接受，使用 normal 找解的結果也不錯，由於染色體長度是 16，我們推測大約使用染色體長度 1/3~1/2 來 crossover 是最有效率的。

4.6 批次傳送

比照現實狀況，在每個 timeslot 產生 request，觀察在許多 timeslot 之下基因演算法與其他演算法的差異。

4.6.1 設定

我們模擬的流程圖如圖 23。並讓網路內沒有背景的流量流過。

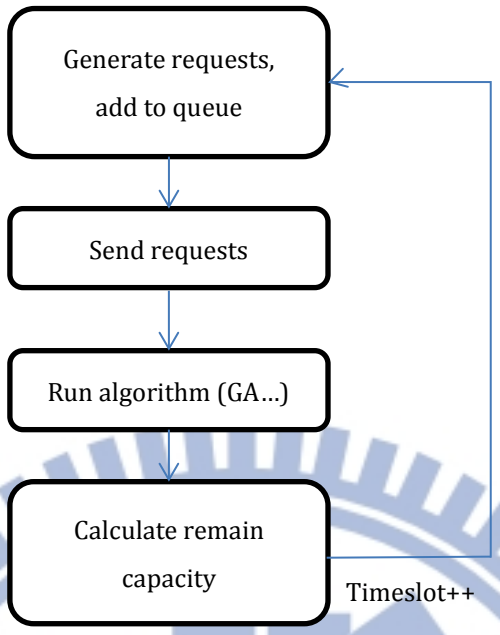


圖 23 Batch 流程圖

Timeslot=100

產生 request：Binomial distribution（P_{gen}）

決定目的 server：Uniform distribution

決定封包大小：Binomial distribution（P_{size}）

$$\text{Load} = \frac{\text{average packet size}}{\text{link capacity}} \times P_{gen}$$

P_{gen} 及 P_{size} 依照 Load 調整大小。

模擬方法有：(1) 4.2 節使用的迴圈決定 drop（以下稱 GA-1）

(2) 4.3 節使用的 GA 決定 drop（以下稱 GA-2）

(3) Sequential；傳送優先順序為封包大小由大到小（seqMax）

(4) Sequential；傳送優先順序為封包大小由小到大（seqMin）

(5) Sequential；傳送優先順序為 server 順序（seqOrder）

(6) LINGO（optimum）

(7) random 選擇交換器 編號，若仍有空間便傳送，反之捨棄

4.6.2 封包大小的影響

封包大小的分布狀況會影響網路路徑配置的狀況，也會影響整體吞吐量及使用能量的表現。表 11 呈現同樣為 Load=5 的時候，不同 P_{gen} 及 P_{size} 造成的差異。

表 11 同 load 不同 P_{gen} 及 P_{size} 的比較

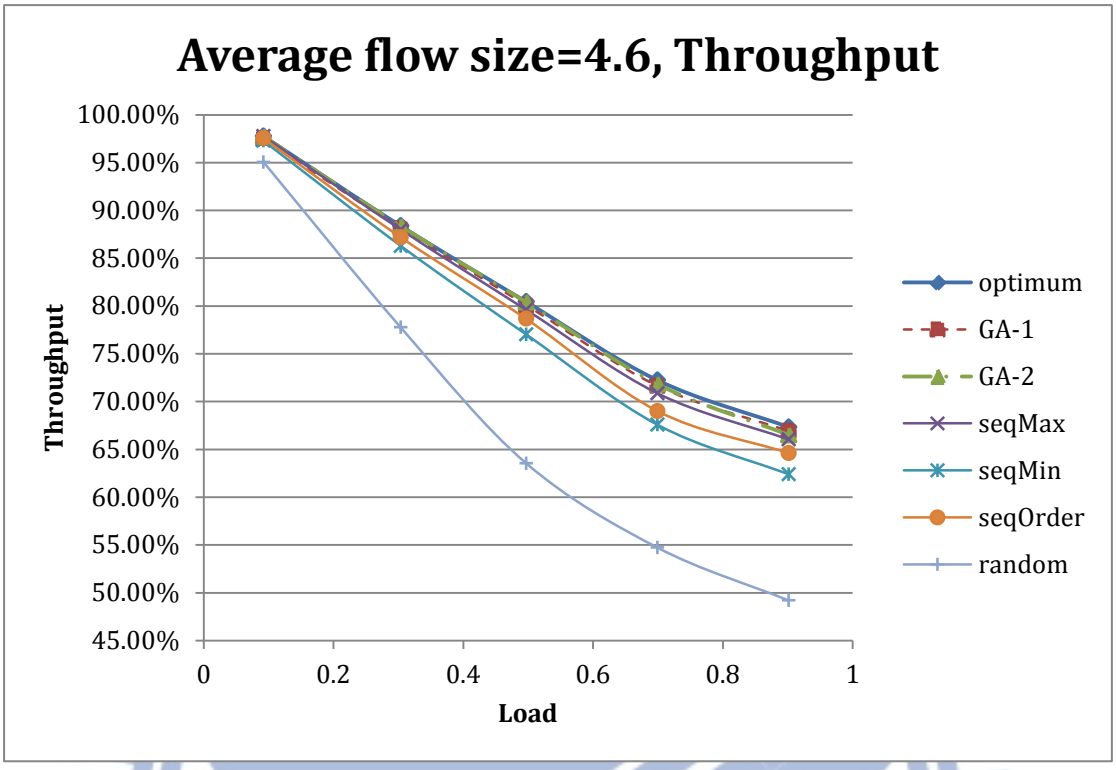
	Average throughput		Average energy	
Load	0.5 P_{gen}=0.5 P_{size} =1	0.5 P_{gen}=0.625 P_{size} =0.75	0.5 P_{gen}=0.5 P_{size} =1	0.5 P_{gen}=0.625 P_{size} =0.75
GA-1	80.99%	78.31%	70.78%	77.07%
GA-2	80.90%	78.36%	69.30%	77.12%
seqMax	80.60%	78.24%	74.37%	79.38%
seqMin	80.60%	73.90%	74.37%	79.57%
seqOrder	80.60%	76.07%	74.37%	79.94%
optimum	80.99%	78.96%	69.08%	76.80%

P_{size}=1 時（封包大小皆為 5），結果會比較好，其原因應為當封包大小都達到鏈結容量上限時，會比較容易安排路徑，不會有封包大小較小的傳送，但大的沒傳送造成的吞吐量降低。然而這種狀況並不符合現實，所以下一小節模擬將會避開封包大小都是最大值的狀況。

4.6.3 不同 load

在此節我們固定 P_{size}，利用控制 P_{gen} 大小來調整 Load。以下分別為 P_{size}=0.9（平均封包大小為 4.6）、P_{size}=0.75（平均封包大小為 4）、P_{size}=0.6（平均封包大小為 3.4）的模擬，並將 100 個 timeslot 的結果平均起來作圖。

1. P_{size}=0.9（平均封包大小為 4.6）



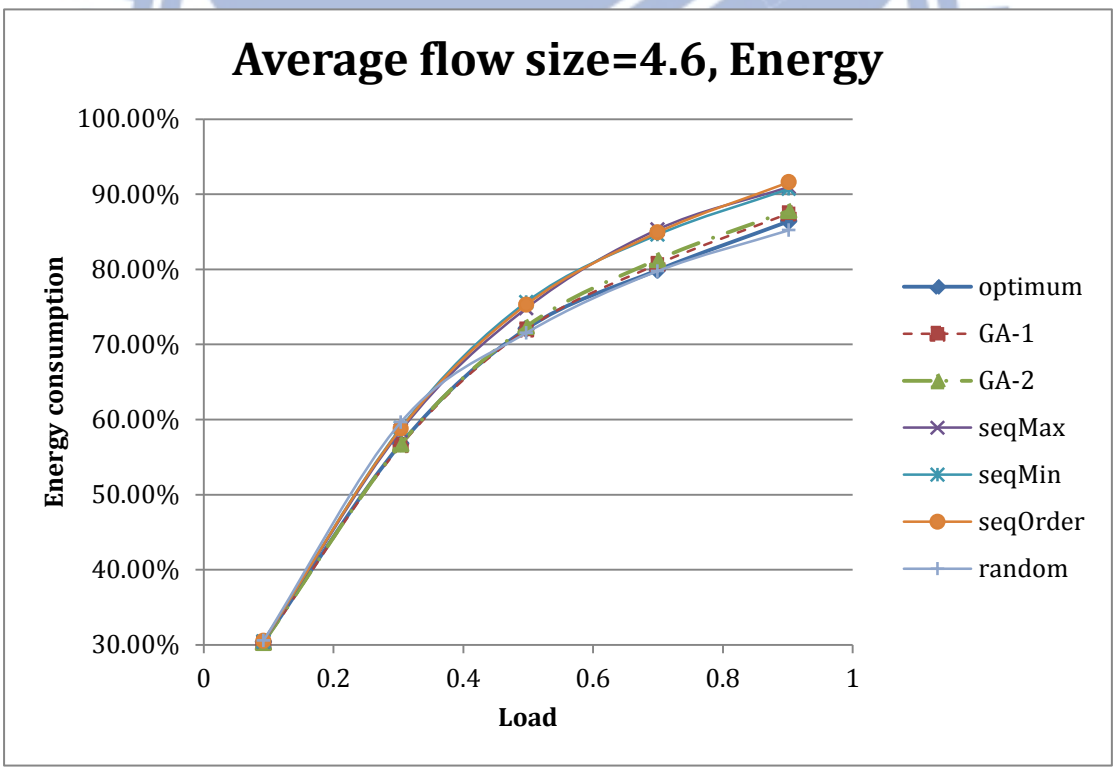
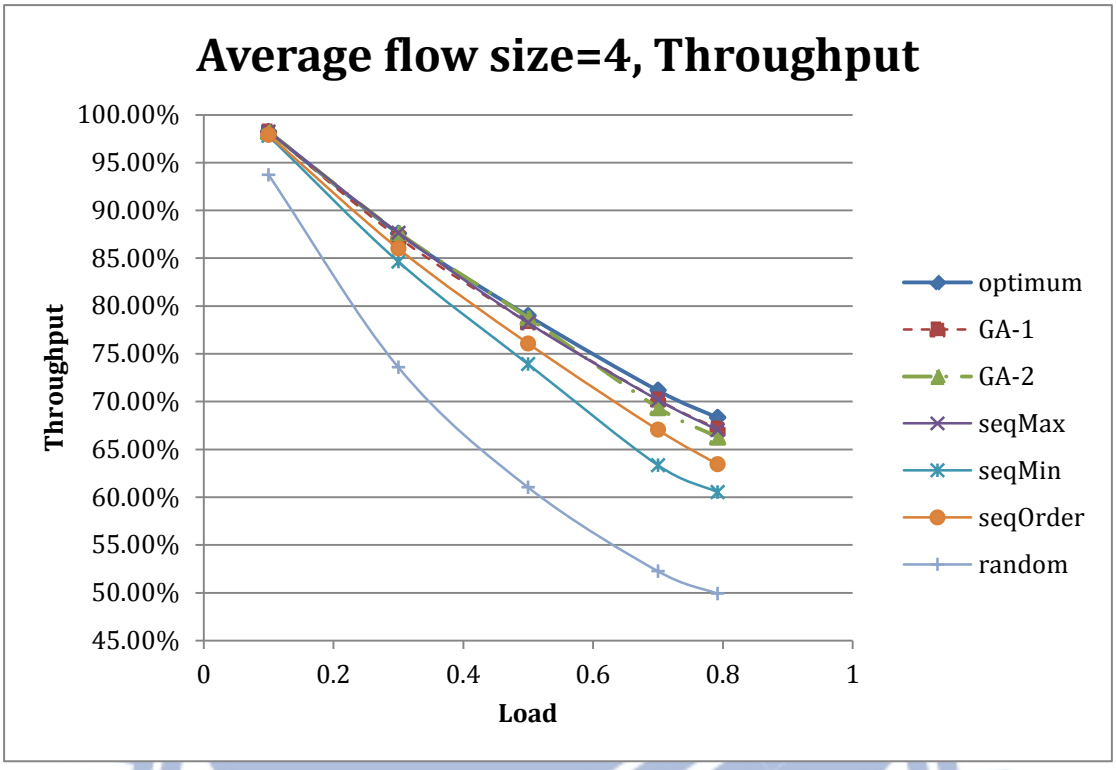


圖 24 Average flow size=4.6

2. P_{size}=0.75（平均封包大小為 4）



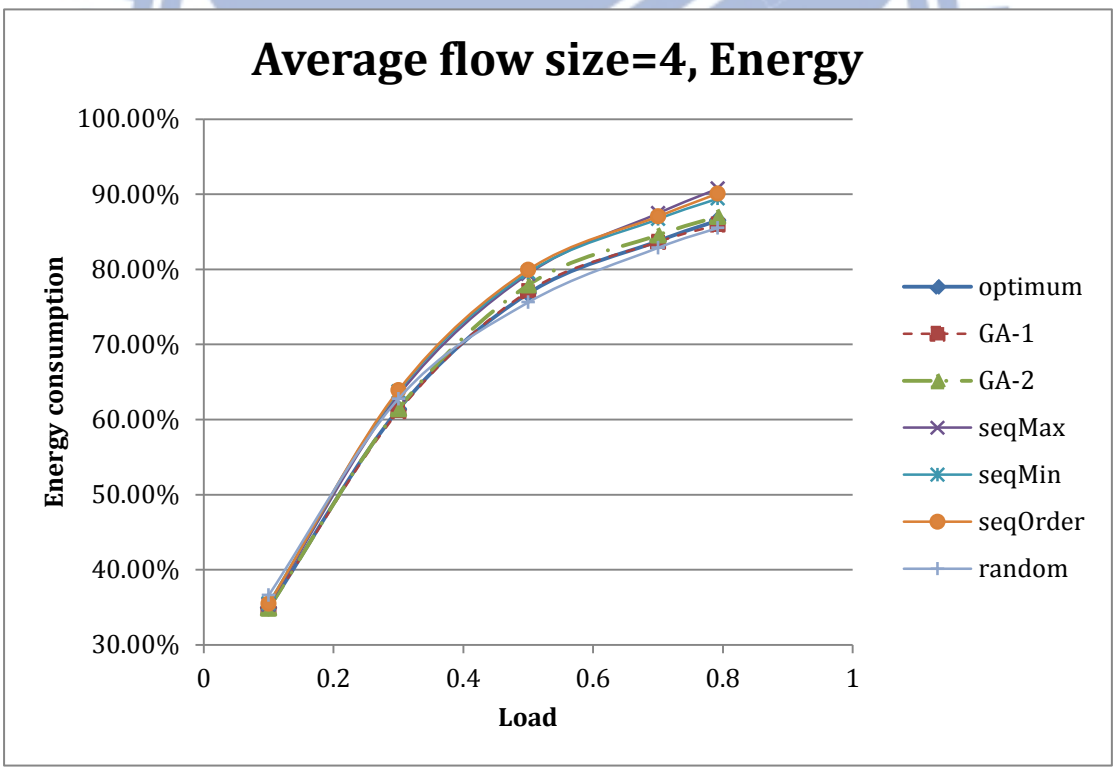
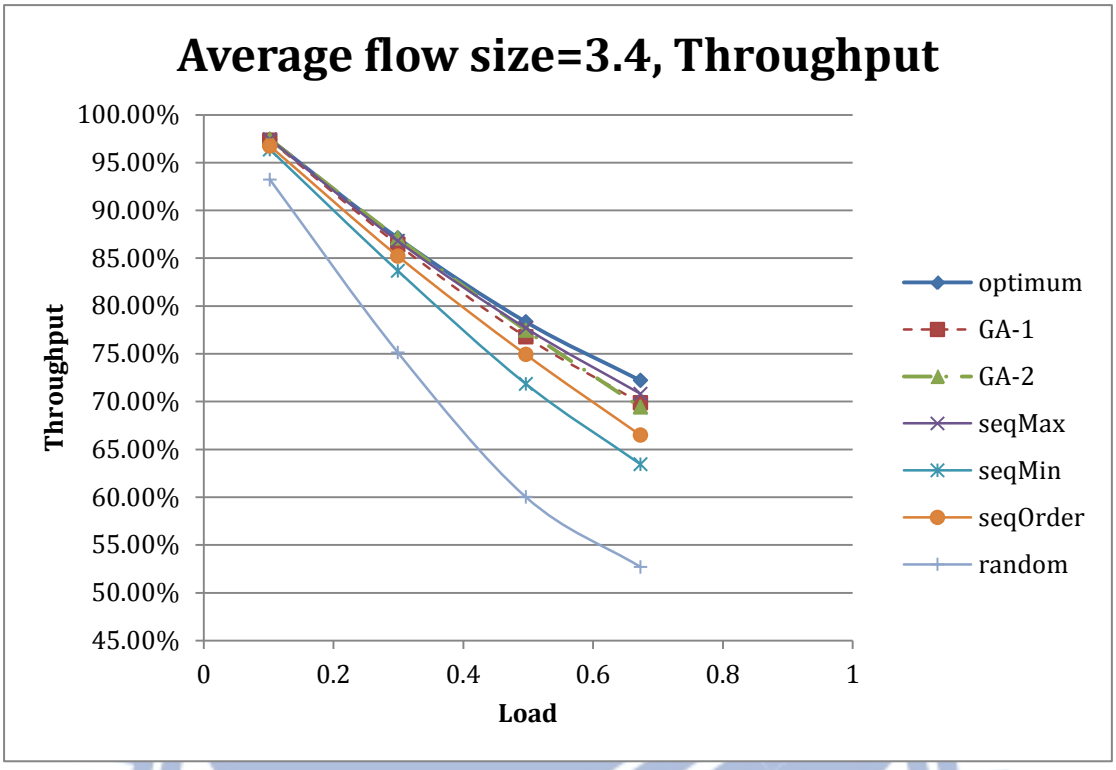


圖 25 Average flow size=4

3. P_{size}=0.6（平均封包大小為 3.4）



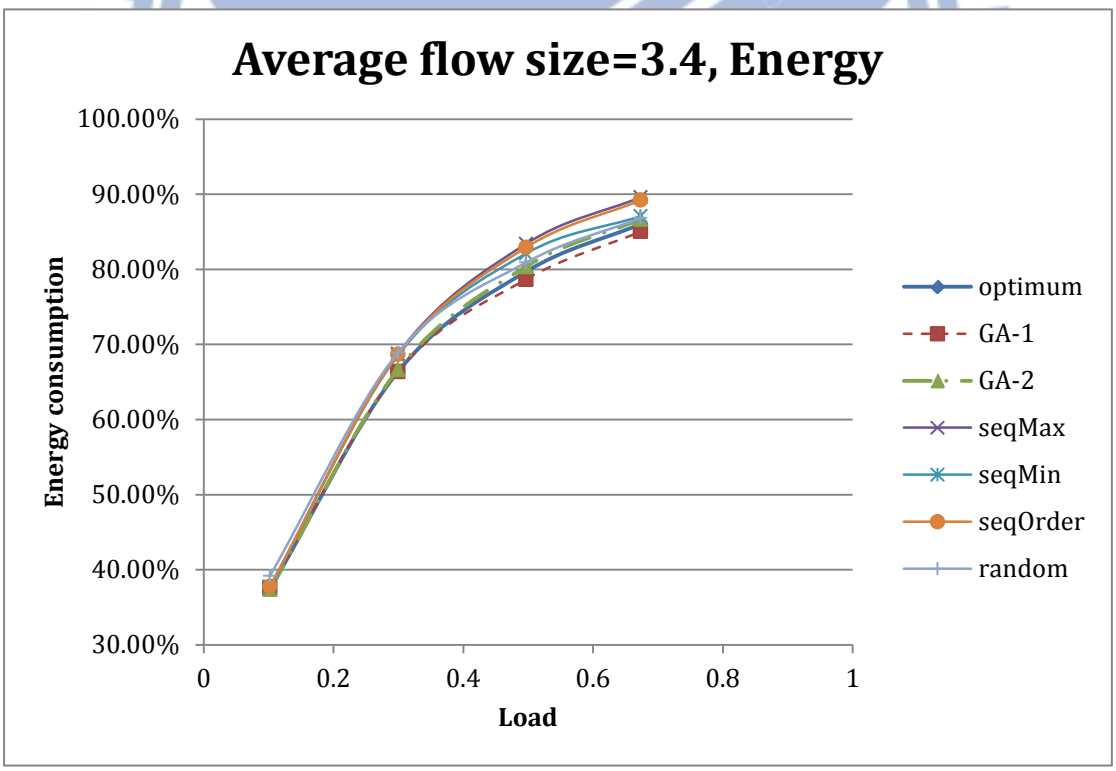


圖 26 Average flow size=3.4

觀察圖 24~26，可發現不論 GA-1 或是 GA-2，在吞吐量及使用能量的表現都與 optimum 的結果十分接近，顯現基因演算法的結果是可信賴的。另外在吞

吐量的圖中，seqMax 的表現也相當好，然而此方法雖然可以達到高吞吐量，但在能量的使用量方面卻沒有這麼好的結果。而 seqMin 及 seqOrder 則是可以明顯看出吞吐量跟使用能量都不及 GA。因此權衡之下還是以 GA 的兩種方法較能達到我們想要的既保證吞吐量也同時降低使用能量的目的。

5 結論

5.1 研究結果

在此論文中，我們提出並改良基因演算法，應用在數據中心內的網路架構 fat-tree 拓樸上，使得我們可以快速的決定網路的路徑分配，並且達到高吞吐量及低耗能的目標。基因演算法是模仿大自然物競天擇的法則來運作的，具有快速且能夠在複雜的解空間中找到良好的解的能力，因此我們將困難的頻寬分配及能量消耗轉換成最佳化問題，並利用基因演算法來達成。

我們針對基因演算法做了許多改善及嘗試，使得基因演算法得到的結果能更切合我們的需求，其中有簡化染色體編碼，縮小解空間以增加搜尋的速度；修改捨棄 request 的機制，得到可接受的吞吐量比率；增加基因分數的計算，快速判斷不同狀況下可能較為優秀的交換器；改良交配及突變方法，從盲目的隨機搜尋轉換成有意義的搜尋等等。而這些嘗試也使演算法能夠符合我們最佳化的問題，在各種 load 下，都可找到近似於最佳解的解並且在合理的時間內收斂。

5.2 未來展望

本論文在實驗結果的最後有與其他方法所得結果相比較，結果雖有較其他方法好，然而因拓樸的規模關係，差距並不是很明顯。因此接下來可嘗試把拓樸放大，測試基因演算法是否可以維持優勢。另外在 GA-2 的方法中，仍是有最後產生的解有 congestion 的可能，將嘗試是否可以進一步把基因分數定義的更精確，並利用在交配跟突變運算中，以期得到更有意義的操作，幫助加快在解空間內搜尋的速度。

6 參考資料

[1] D. Grunwald, P. Levis, K. Farkas, C. M. III, and M. Neufeld. Policies for Dynamic Clock Scheduling. In *OSDI*, 2000.

[2] C. Patel, C. Bash, R. Sharma, M. Beitelmam, and R. Friedrich. Smart Cooling of data Centers. In *Proceedings of InterPack*, July 2003.

[3] A. Greenberg, J. Hamilton, D. Maltz, and P. Patel. The Cost of a Cloud: Research Problems in Data Center Networks. In *ACM SIGCOMM CCR,* January 2009.

[4] U.S. Environmental Protection Agency. Report to Congress on Server and Data Center Energy Efficiency. http://tinyurl.com/2jz3ft

[5] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. Mckeown, "ElasticTree: Saving Energy in Data Center Networks," *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, 2010.

[6] P. Mahadevan, P. Sharma, S. Banerjee, and P. Ranganathan, "A Power Benchmarking Framework for Network Devices," *Proceedings of IFIP Networking,* May 2009.

[7] C. E. Leiserson, "Fat-Trees: Universal Networks for Hardware-Efficient Supercomputing," *IEEE Transactions on Computers*, vol. 34, no. 10, October 1985, pp. 892-901.

[8] M. Al-Fares, A. Loukissas, and A. Vahdat. A Scalable, Commodity Data Center Network Architecture. In *ACM SIGCOMM*, pages 63-74, 2008.

[9] Holland, J. H., Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, 1975.

[10] 林昇甫、徐永吉，「基因演算法及其應用」

[11] Z. Yuanping, M.Zhengkun, and X. Minghai, "Dynamic load balancing based on roulette wheel selection," *Proc. Int. Conf. Communications, Circuits and Systems,* vol. 3, pp.1732-1734, 2006.

[12] D. Wicker, M. M. Rizki, and L. A. Tamburino, "The multi-tiered tournament selection for evolutionary neural network synthesis," *Proc. Int. Conf. Combinations of Evolutionary Computation and Neural Networks*, pp.207-215, 2000.

[13] W. M. Spears, K. A. De Jong, T. Back, D. B. Fogel, and H. deGaris, "An overview of evolutionary computation,"*Proc. Conf. Machine Learning*, 1993.

[14] D. Beasley, D. R. Bull, and R. R. Martin, "An overview of genetic algorithms: Part 1, Fundamentals", *University Computing*, vol.15, no. 2, pp. 58-69, 1993.

[15] K. Y. Lee and P. S. Mohamed, "A real-coded genetic algorithm involving a hybrid crossover method for power plant control system design," *Proc. Int. Conf. Evolutionary Computation*, pp. 1069-1074, 2002.