一個基因演算法在瓶頸選擇內部節點最小生成樹問題上

A genetic algorithm for the bottleneck selected-internal minimum spanning tree problem

Chien Yuan Ho[1], Yen Hung Chen[2]

Department of Computer Science,
University of Taipei, Taiwan
h10292@hotmail.com[1] , yhchen@utaipei.edu.tw[2]

摘要

本文探討一個組合最佳化問題：瓶頸選擇內部節點最小問題（bottleneck selected-internal minimum spanning tree problem）。給定一個無向完全圖$G=(V,E)$，一個非負數的權重函數在邊上及一個節點的集合$R\subset V$，選擇內部節點生成樹為一棵生成樹連通V內的所有節點，但是在R內的節點不能是樹葉（$|V\backslash R|\geq 2$）。瓶頸選擇內部節點最小生成樹問題定義為找出一棵選擇內部節點最小生成樹使得樹上最大邊的權重要最小。此研究已被證明為NP-Complete，此問題可應用在廣播資源配置(broadcast resource allocation)。本研究設計一個基因演算法來解決問題，並透過隨機產生的亂數數據模擬測試，實驗結果顯示我們的基因演算法所得的解與最佳解的倍率大約為1.0至1.2之間，最高到1.15倍。在執行效率上，我們的基因演算法所用之時間平均為2至3毫秒（ms）。

1 緒論

在組合最佳化問題中，瓶頸問題（bottleneck problems，或稱為 min-max problems）經常出現在許多通訊網絡的設計（communication network design)、作業研究中企業或工廠上的倉儲（warehouse）及配送中心等服務規劃（service planning）、交通運輸網路上的緊急設施規劃（emergency facility planning）、工作排程（scheduling）、資源管理（resource management）、容量規劃（capacity planning）等應用上[1,2,5,6,7,11,14,15,17,20,21]。這類瓶頸問題的顯著特性是會存在一個最差的（worst case）值或條件稱為 bottleneck 在整個（網路或企業）環境中。我們的目的是希望找到最差的情況（bottleneck）要最小。一些非常著名的瓶頸問題有中心點問題（Center problem）[11]、瓶頸最小生成樹問題（Bottleneck minimum spanning tree problem）[4]、瓶頸最小 Steiner 樹（Bottleneck minimum Steiner tree problem）[3,8]及瓶頸最小旅行推銷員問題（Bottleneck minimum traveling salesman problem，BTSP）[4,10,19]等。這些瓶頸問題有許多的多項式時間演算法可以找到問題的最佳解或近似解 [3,4,8,10,19]。

一個應用在群播繞境(Multicast routing) [1,2,6,7,17]非常著名的問題：斯坦納樹問題(Steiner tree problem)。給定一個無向圖$G=(V,E)$，一個非負數的權重函數在邊上及一個節點集合$R\subseteq V$，一棵斯坦納樹（Steiner Tree）定義為圖上的一棵樹連通 R 內的所有節點。斯坦納樹問題為在圖上找一棵權重加總最小的斯坦納樹 [12]。基於 Sensor network allocation 應用，謝孫源教授與楊仕丞定義了斯坦納樹問題的變形問題：選擇內部節點斯坦納樹問題（selected-internal Steiner tree problem） [16]。給定無向完全圖 $G=(V,E)$，一個非負數的權重函數在邊上及有兩個節點集合R與R'，$R\subseteq V$且$R'\subset R$。選擇內部節點斯坦納樹問題(SISTP)為在圖上找一棵權重加總最小的斯坦納樹連通 R 內所有節點但要求 R'內的節點不能是樹葉[17]。Li 等人提出了一個生成樹版本的選擇內部節點斯坦納樹問題：選擇內部節點最小生成樹問題（selected-internal minimum spanning tree problem） [18]。給定一個無向完全圖 $G=(V,E)$，一個非負數的權重函數（cost function）在邊上及一個節點的集合$R\subset V$，選擇內部節點生成樹定義為一棵生成樹連通 V 內的所有節點，但是在 R 內的節點不能是樹葉（$|V\backslash R|\geq 2$）。選擇內部節點最小生成樹問題(SIMSTP)定義為找出一棵選擇內部節點最小生成樹其權重加總要最小 [18]。然而這些問題都為 NP-Hard [9,16,18]。Ho，Kuo 及 Chen [13]提出了一個瓶頸問題版本的選擇內部節點最小生成樹問題：瓶頸選擇內部節點最小生成樹問題（bottleneck selected-internal minimum spanning tree problem）。瓶頸選擇內部節點最小生成樹問題（BSIMSTP）定義為找出一棵選擇內部節點生成樹使得樹上最大邊的權重要最小 [13]。他們證明 BSIMSTP 為 NP-hard 接著設計了一個啟發式演算法（heuristic algorithm）來解決此問題，並模擬在一些亂數產生的資料上。實驗結果所得在執行效率上，時間平均為 1 至 5 毫秒（ms）。所用的啟發式演算法是透過 $V\backslash R$ 內挑出任兩節點分別為起點 v_s 及終點 v_d，接下來找出一個 path 從起點 v_s 到終點 v_d 並經過 V 內所有的節點一次，形成節點集合 V 的一個漢米爾頓路徑(Hamiltonian path) [4]（可透過 BTSP 的 2 倍

近似演算法 [19]）後找出 BSIMSTP 的一個合法解(feasible solution)。一個很著名的啟發式演算法:基因演算法(genetic algorithm)。此演算法是模擬自然進化過程的隨機搜尋法，在組合最佳化問題中能夠於短時間內提供有效並不錯的解。在本論文中，我們設計一個基因演算法解決 BSIMSTP，並與正確演算法（exact algorithm）及 Ho，Kuo 及 Chen 所提出的啟發式演算法 [13]（簡稱 Ho 的啟發式演算法）進行比較。經過實驗程式模擬之後顯示基因演算法所得的解與最佳解的倍率（我們的演算法所得出的解的權重除以最佳解的權重）在 1.0 至 1.2 之間，最高是 1.15 倍。而執行效率則平均約為 2 至 3 毫秒。相比於 Ho 的啟發式演算法，基因演算法穩定很多，不受內部節點數多寡影響，對於內部節點數越少的情況，基因演算法能提供比啟發式演算法更好的時間效率，而在整體近似倍率上也能更接近於最佳解。

本論文第二章我們將描述我們的基因演算法來解決 BSIMSTP。第三章利用我們的基因演算法對一些亂數產生的資料進行模擬(simulation)測試，藉此結果說明基因演算法與 Ho 的啟發式演算法及正確演算法相比後的結果。最後在第四章，我們說明 BSIMSTP 的未來研究方向。

2 基因演算法應用於 BSIMSTP 上

本章中，我們設計一個基因演算法來解決 SIMSTP，找出其合法解。底下小節我們敘述我們的基因演算法的設計過程。

2.1 編碼設計

在之前的研究中[13]，我們已知漢米爾頓路徑問題的解可成為我們問題的合法解，為了解決我們的問題，我們將基因演算法建立在漢米爾頓路徑問題上。因此，我們將無向圖 G 中的個節點加以編號，編碼過程中隨機選出兩個非內部的節點作為編碼字元串的頭尾，其餘節點編號則隨機分配到字元串內部，如此便完成一組染色體。

2.2 交配模式與突變率設計

基因演算法常見交配方式有單點法(single-point crossover)、雙點法(two-point crossover)及均勻法(uniform crossover)。針對問題我們選擇單點法做為主要的交配模式基礎。然而我們對此模式做一點小修改，為確保染色體串列在交配後不會使非內部節點移到頭尾與目標條件衝突，我們挑出非內部節點優先對頭尾進行交換後再對染色體內部做單點法交配。突變率上，我們設定各組染色體在交配後有10%的機率會隨機挑選染色體內部任兩點交換以增加演化的多樣性。

2.3 適應函數設計

設定各組基因編號串連的最大邊的權重為 $W_i(i=1,2,...,基因數)$，對於本問題我們將定義適應函數為下列式子

$$F = [\ (E_0 - W_i) + 1\ ]*(\ E_0*E_n\ ) - S_i$$

E_0 為原無向圖最大邊；E_n 為邊數，因為所求為生成樹，故此符號相當於 $V-1$(總節點數減一)；S_i 為各組基因編號串總邊長權重總和。此式同時能確保適應度恆為正值，並讓最各組基因中最大邊權重的差異變大。

2.4 整體架構設計

為了讓基因演算的過程能較快的速度往最佳解前進，我們的參數控制如下所示：

我們描述此演算法如下：

輸入：一個無向完全圖 $G=(V,E)$，每個邊上有非負數之權重函數(需滿足三角不等式)，以及一內部節點集合 $R \subset V$（$|V \backslash R| \geq 2$）。

輸出：建構一個在 G 內生成樹 T 使得每個 R 內的節點必須為此生成樹的內部節點。

Step1. 參數設定，我們設母體大小(population)即染色體總數設為 10 組、演化代數為 50 代並額外設定一組基因作為菁英策略(Elitism strategy)。

Step2. 產生初代母體基因，依 2.1 小節的編碼設計所示完成 10 組染色體。

Step3. 10 組母體依前後分為五對交配，交配模式參照 2.2 小節說明，交配產生的子代再以 10%的突變機率來增加染色體的變異組合。

Step4. 經由 Step3 的操作後，以 2.3 小節的適應函數判斷子代的適應能力，各子代依其適應度在整體中所佔比例以同等機率做為下一代母體的選擇，其中適應度最高的子代將複製一份做為菁英讓演化必定朝最佳解前進。

Step5. 重覆 Step 3 和 Step 4 直到演化代數完成。最後產出的菁英染色體即為我們問題透過基因演算法所得的解。

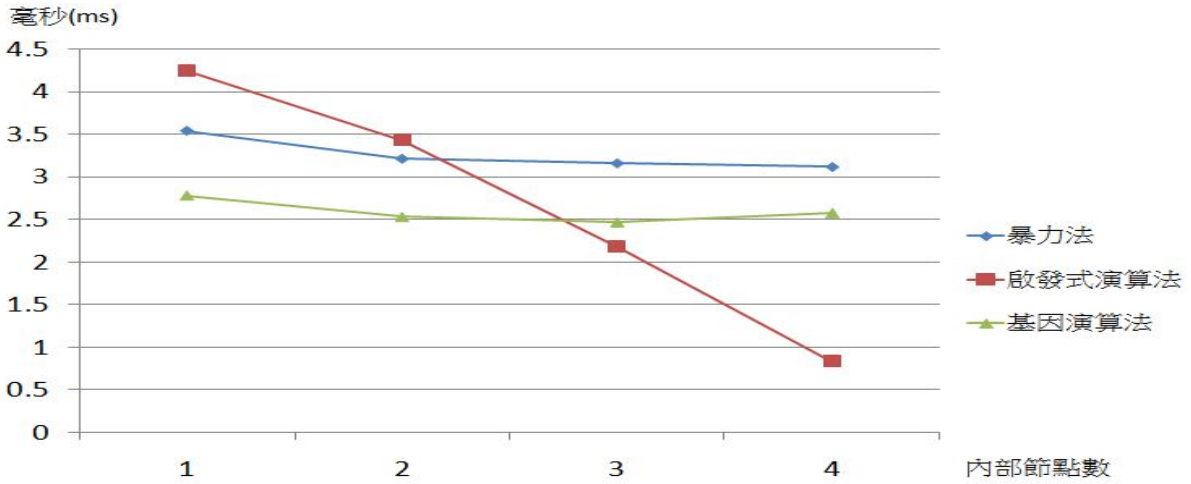

圖 1：在輸入的圖為 6 個節點下，基因演算法、Ho 的啟發式演算法與暴力法之耗時比較（單位：毫秒）。每個點的值是執行 50 組資料後的平均值。

3 實例驗證

本章中，我們將針對第二章所設計的演算法來進行實際的程式模擬。我們藉由 brute force 的正確演算法(exact algorithm)、Ho 的啟發式演算法 [13]與我們設計的基因演算法進行時間效率的分析，我們也會觀察基因演算法所產生的解的權重與最佳解的權重之間的倍數（近似率）關係。近似率(ratio)是拿我們設計的基因演算法所得出的解的權重除以最佳解的權重。Ho 的啟發式演算法是利用 BTSP 之 2 倍的近似演算法[19]改良後得到。本次模擬實驗使用 Eclipse 撰寫 Java 程式，作業系統平臺為 64 位元 Microsoft Windows 7 SP1。處理器使用 Intel Core i5-3450，時脈 3.1GHz，記憶體為 8GB。分別以亂數產生 6 個節點與 7 個節點之完全圖（權重符合 metric）並用二維陣列儲存無向完全圖之節點及其權重。第一小節中，我們模擬的情形是透過亂數產生器產生 6 個節點的完全圖。第二小節中，我們模擬的情形是透過亂數產生器產生 7 個節點的完全圖。

3.1 實測節點數為 6 時的案例

本小節中我們實測當輸入的圖有 6 個節點時，基因演算法、Ho 的啟發式演算法與正確演算法所產生的解之間的關係。正確演算法是利用暴力法窮舉所有的生成樹得到 SIMSTP 的最佳解。模擬的情形再分內部節點設定為 1 到 4 個節點的情形，每個實驗分別用亂數產生 50 個隨機的完全圖並將執行後的結果取平均。

為了說明我們的演算法所產生的解與最佳解之間並不會相距太遠，表 1.分別列出了啟發式演算法與基因演算法所產生出的解與最佳解之間的近似率。實驗結果顯示這兩個演算法的近似率十分接近 1。透過基因演算法在節點數為 6 時可得到相當好的結果。雖然我們未能以數學證明我們的解距離最佳解的倍率，然而實驗數據顯示我們的解不失為一個好的結果。

內部節點數	1	2	3	4
近似率(Ho 的啟發式演算法)	1.058	1.072	1.063	1.097
近似率(基因演算法)	1.087	1.054	1.043	1.033
標準差(Ho 的啟發式演算法)	0.108	0.095	0.095	0.103
標準差(基因演算法)	0.114	0.066	0.063	0.068

表 1. BSIMSTP 的近似率模擬結果(找出最佳解分別與基因演算法及啟發式演算法的解間的差異)，針對輸入為 6 個節點時情形。近似率分別為基因演算法與 Ho 的啟發式演算法的解的權重除以最佳解的權重。每個數據的值是執行 50 組資料後的平均值。

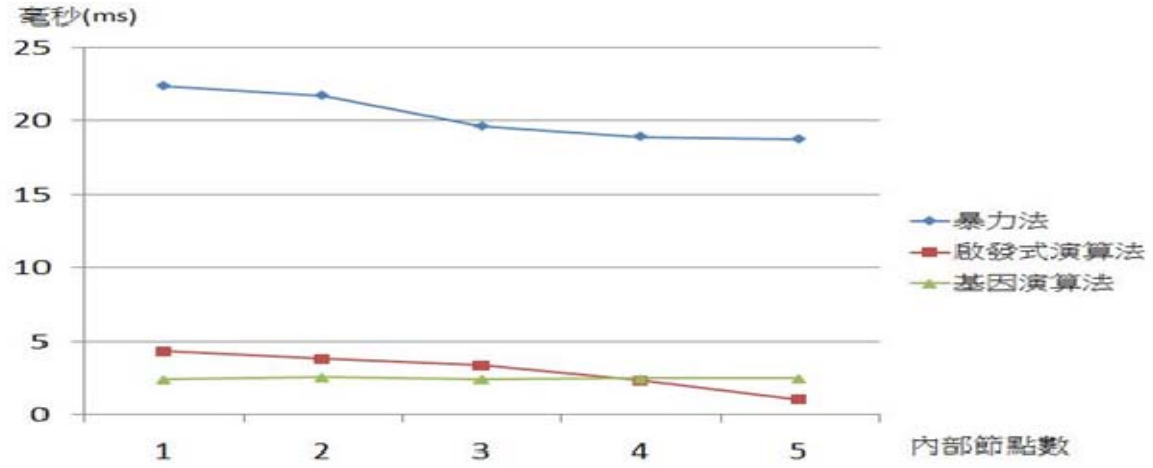

圖 2：在輸入的圖為 7 個節點下，基因演算法、Ho 的啟發式演算法與暴力法之耗時比較（單位：毫秒）。每個點的值是執行 50 組資料後的平均值。

在時間分析上，我們同樣用分析近似率的圖來觀察基因演算法、Ho 的啟發式演算法與正確演算法在執行時的效率。圖 1.列出了三個演算法在效率上的比較結果。在 6 個節點時三種演算法執行時間平均差異不大，橫座標是內部節點的個數，縱座標是執行的時間(毫秒)。在內部節點為 1 時，正確演算法在執行上平均需花 3.535 毫秒、啟發式演算法平均需花 4.244 毫秒，基因演算法平均僅花 2.777 毫秒。在內部節點為 4 時，正確演算法平均需花 3.115 毫秒、啟發式演算法平均需花 0.835 毫秒，而基因演算法平均花 2.576 毫秒。實驗結果顯示，啟發式演算法會隨內部節點多寡影響執行速度，然而基因演算法則平均落在 2.4 至 2.8 毫秒之間。

3.2 實測節點數為 7 時的案例

本小節中我們實測當輸入的圖有 7 個節點時，基因演算法、Ho 的啟發式演算法與正確演算法產生的解之間的關係。內部節點分別設定為 1 到 5 個節點，每個實驗亦是分別用亂數產生 50 個隨機的完全圖並將執行後的結果取平均。

表 2.分別列出了基因演算法及啟發式演算法所產生出的解與最佳解之間的近似率(ratio)。實驗結果顯示啟發式演算法的近似率較平均但普遍都比基因演算法大。基因演算法較能夠更精確的接近正確演算法的結果。

在時間分析上，我們同樣用分析近似率的圖來觀察基因演算法、Ho 的啟發式演算法與正確演算法在執行時的效率。圖 2.列出了三個演算法在效率上的比較結果。在 7 個節點時，基因演算法與 Ho 的啟發式演算法的時間明顯低於正確演算法，這應該是很直覺的，因為正確演算法的時間複雜度成指數成長。橫座標是內部節點的個數，縱座標是執行的時間(毫秒)。在內部節點為 1 時，正確演算法平均需花 22.395 毫秒，啟發式演算法平均需花 4.365 毫秒，而基因演算法需要 2.432 毫秒。在內部節點為 5 時，正確演算法平均需花 18.755 毫秒，啟發式演算法平均需花 1.058 毫秒，而基因演算法平均 2.517 毫秒。實驗結果顯示，基因演算法較不受內部節點多寡影響。

4. 未來研究方向

本研究探討瓶頸選擇內部節點最小生成樹問題(BSIMSTP)，我們對此問題設計一個基因演算法來找出問題的合法解。透過實驗顯示基因演算法的倍率大約為1.0至1.2之間，最高到1.15倍。基因演算法在時間上相對穩定很多，不受內部節點數多寡影響，內部節點數越少的情況，基因演算法能提供比啟發式演算法更好的效率，而在整體近似倍率上也能提供更相近於正確演算法的最佳解。未來我們希望能找到一個更好的適應函數來設計一個新的基因演算法，其時間能夠更有效率，近似率也能更接近最佳解。

Acknowledgements

This work was supported in part by the National Science Council of the Republic of China under Contract NSC 102-2221-E-133-002-3 。 Corresponding author: Yen-Hung Chen。

內部節點數	1	2	3	4	5
近似率(Ho的啟發式演算法)	1.122	1.111	1.116	1.115	1.161
近似率(基因演算法)	1.151	1.107	1.116	1.061	1.043
標準差(Ho的啟發式演算法)	0.101	0.102	0.117	0.115	0.122
標準差(基因演算法)	0.118	0.112	0.119	0.076	0.065

表 2. BSIMSTP 的近似率模擬結果(找出最佳解分別與基因演算法及啟發式演算法的解間的差異)，針對輸入為 7 個節點時情形。近似率為我們兩個演算法的解的權重除以最佳解的權重。每個數據的值是執行 50 組資料後的平均值。

Reference

[1] M. Charikar, J. Naor, and B. Schieber, Resource optimization in QoS multicast routing of real-time multimedia, *IEEE/ACM Transactions Networking*, Vol. 12**,** pp. 340–348, 2004.

[2] X. Cheng and D.Z. Du, *Steiner Tree in Industry*, *Kluwer Academic Publishers*, Dordrecht, Netherlands, 2001.

[3] C. Chiang, M. Sarrafzadeh, and C.K. Wong, Global router based on Steiner min-max trees, *IEEE Transaction on Computer-Aided Design*, Vol. 9, pp. 1318–1325, 1990.

[4] T.H. Cormen, C.E. Leiserson, R.L. Rivest and C. Stein, *Introduction to Algorithm*, 2nd edition, MIT Press, Cambridge, 2001.

[5] M.S. Daskin, *Network and Discrete Location: Models Algorithms and Application*s, Wiley, New York, 1995.

[6] D.Z. Du, J.M. Smith and J.H. Rubinstein, *Advances in Steiner tree*, Kluwer Academic Publishers, Dordrecht, Netherlands, 2000.

[7] D.Z. Du and X. Hu, *Steiner Tree Problems in Computer Communication Networks*, World Scientific Publishing Company, 2008.

[8] C.W. Duin and A. Volgenant, The partial sum criterion for Steiner trees in graphs and shortest paths, E*uropean Journal of Operations Research*, Vol. 97, pp. 172–182, 1997.

[9] M.R. Garey, R.L. Graham, and D.S. Johnson, The complexity of computing Steiner minimal trees, *SIAM Journal of Applied Mathematics*, Vol 32, pp. 835–859, 1997.

[10] R.S. Garfinkel and K.C. Gilbert, The bottleneck travelling salesman problem Algorithms and probabilistic analysis, *Journal of the ACM*, Vol. 25, pp. 435–448, 1978.

[11] S.L. Hakimi, Optimal locations of switching centers and the absolute centers and medians of a graph, *Operations Research,* Vol 12, pp. 450–459, 1964.

[12] S.L. Hakimi, Steiner's problem in graphs and its implications, *Networks*, Vol. 1, pp. 113-133, 1971.

[13] C.Y. Ho, Y.T. Kuo and Y.H. Chen, A heuristic algorithm for the bottleneck selected-internal minimum spanning tree problem, *The 30th Workshop on Combinatorial Mathematics and Computational Theory*, pp. 75–80, 2013.

[14] D.S. Hochbaum and D.B. Shmoys, A unified approach to approximation algorithms for bottleneck problems, *Journal of the ACM*, Vol 33, pp. 533–550, 1986.

[15] D.S. Hochbaum and A. Pathria, Generalized p-center problems: complexity results and approximation algorithms, *European Journal of Operational Research*, Vol 100, pp. 594–607, 1997.

[16] S.Y. Hsieh and S.C. Yang, Approximating the selected-internal Steiner tree, *Theoretical Computer Science*, Vol. 381, pp. 288–291, 2007.

[17] F.K. Hwang, D.S. Richards and P. Winter, *The Steiner Tree Problem*, Annuals of Discrete Mathematics, Vol. 53, Elsevier Science Publishers, Amsterdam, 1992.

[18] X. Li, F. Zou, Y. Huang, D. Kim and W. Wu, A better constant-factor approximation for selected-internal Steiner minimum tree, *Algorithmica*, Vol. 56, pp. 333–341, 2010.

[19] R.G. Parker and R.L. Rardin, Guaranteed performance heuristics for the bottleneck traveling salesman problem. *Operations Research Letters*, Vol. 2, pp. 269–272, 1984.

[20] C.S. ReVelle and H.A. Eiselt, Location analysis: A synthesis and survey, European Journal of Operational Research, Vol. 165, pp. 1–19, 2005.

[21] A. Tamir, Improved complexity bounds for center location problems on networks by using dynamic data structures, SIAM Journal of Discrete Mathematics, Vol 1, 377–396, 1988.