Heuristic Algorithms For Clustered Steiner Minimum Tree Problem

余柏頡 吳邦一
Po-Chieh Yu, Bang Ye Wu
國立中正大學
資訊工程學系暨研究所
ru5017dy@gmail.com, bangye@cs.ccu.edu.tw

摘要

這篇論文主要是針對新問題分群斯坦納樹問題(Clustered Steiner tree problem 簡稱 CluSteiner)提出三個啟發示演算法(heuristic algorithm)，CluSteiner 問題是斯坦納樹問題(Steiner tree problem) 的變形，而這是個實作在度量圖(metric graph)上的問題，最小斯坦納樹問題(Steiner minimum tree problem 簡稱 SMT)問題是被廣泛研究的 NP-hard 問題，而在 CluSteiner 問題裡求出最小花費也已經被證明是 NP-hard 問題。

由於 SMT 問題是 NP-hard 問題，所以求解的時間會因為斯坦納點(Steiner vertex)多寡成指數倍影響，最小 CluSteiner 問題也會因給的 Steiner vertex 數量多寡而造成求解時間成指數倍成長或減少，在這篇論文裡我們提出的三個 heuristic algorithms 能在短時間內，給出不錯的解答。

1. 前言

分群斯坦納樹問題(clustered Steiner tree problem 簡稱 CluSteiner)是斯坦納樹問題(Steiner tree) 的變形，是由吳邦一(Bang Ye Wu)提出的新問題[4]。最小斯坦納樹問題(Steiner minimum tree problem 簡稱 SMT)是 Steiner tree problem 裡被廣泛研究的一個題目，SMT 是給一個簡單無向圖(simple undirected graph) $G = (V, E, c)$和需求點集合(required vertex set)$R \subseteq V$，最後的結果，斯坦納樹(Steiner tree)會是一個 G 的子圖(subgraph)，這子圖包含所有 required vertices 且這子圖要為連通(connected)與無迴路(acyclic)，而這子圖所有邊(edge)上的花費(cost)加總起來要是最少的。SMT 已經是被廣泛研究的 NP-hard 題目，不只是在社會網路領域(Social Network)也在通訊控制(Communication Control)領域[3]也有研究，還有很多領域有研究，所以這問題有很多變形，光是圖就有分實作在一般度量圖(metric graph)上還有歐幾里得空間上的歐幾里得斯坦納樹(Euclidean Steiner minimum trees 簡稱 Euclidean SMT)，更多有關 SMT 詳細介紹可以在[6]找到。

在這篇論文裡，我們是針對 SMT 的變形 CluSteiner 問題提出三個啟發示演算法(heuristic algorithm)，最小 CluSteiner 問題也是 NP-hard 問題，相關證明在[4]。我們簡述 CluSteiner 問題，給一個 metric graph $G = (V, E, c)$還有 required vertex set R 的 partition $\mathcal{R} = \{R_1, R_2, \ldots, R_k\}$，設$T_i$是包含$R_i$的最小 Steiner tree，出來的結果將會是由 k 個不相交的(disjoint)樹T_i組成的 G 的子圖$\mathcal{T}$，這個子圖$\mathcal{T}$一樣是 Steiner tree，且$\mathcal{T}$邊上所有 cost 加起來要是最少，此$\mathcal{T}$稱為 clustered Steiner minimum tree。我們將在下一節給出更完整的定義以及圖例。

這三個 heuristic algorithms 是由[7]啟發的，clustered Steiner minimum tree 如前面所說是 NP-hard 問題，所以如果要求得最加解，所需要的時間將會依 Steiner vertex 的多寡成指數倍成長，而我們提出的演算法主要是能在短時間內得到還不差的結果，這與最佳解演算法比起來時間縮短了非常多，而這三個演算法的詳細做法以及結果都將會在後面的內容介紹。

在這篇論文，將在下一節詳細說明一些使用的符號以及定義。在第三節則會開始講我們三個演算法的實作方法，還有虛擬程式碼(pseudo code)。在第四節，我們則會給我們的實驗結果。在最後一節將會對這整個研究做結論與討論。

2. 符號與定義

在這一節我們給一些定義以及會用到的符號，我們給的圖 $G = (V, E, c)$，V為點集合，E為邊集合，c為邊上的 cost。G為 undirected graph，且為 metric graph，metric graph 需要滿足下列條件：此圖必須為完全圖(complete graph)而所有邊上的 cost 為非負的(nonnegative)且任三邊都要滿足三角不等式。R為點集合，裡面放的點稱作 required vertex，$R \subseteq V$，其餘不在 R 裡面的點稱作 Steiner vertex，在此我們給出論文[4]的問題定義。

Definition 1. 設T為有連到 R 所有點的樹，也就是說，$R \subseteq V(T)$，R 的 *local tree* 是 T 中包含 R 點的最小子樹。

根據定義，local tree 上所有的 leaf 都要是屬於R。

Definition 2. 我們定義$\mathcal{R} = \{R_i \mid 1 \le i \le k\}$為 R 的 partition，如果我們對 R 的一個 Steiner tree T 稱作 *clustered Steiner tree*，表示這 T 裡所有包含$R_i \in \mathcal{R}$的 local trees 互相互斥(disjoint)，也就是說，會有一個 cut set $C \subset E(T)$且$|C| = k - 1$使得$T - C$後每一個 component 都是R_i的 Steiner tree T_i，$1 \le i \le k$。

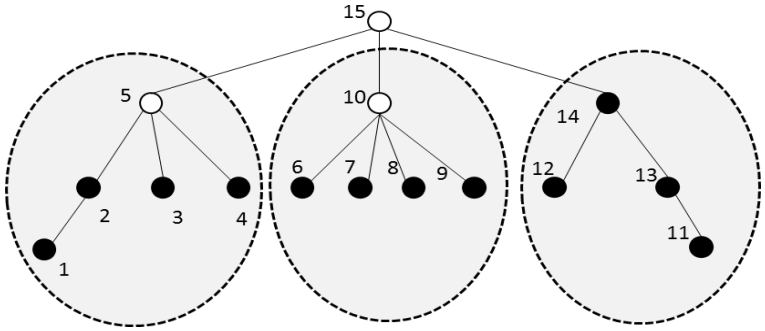

圖 1-1. Minimum CluSteiner 示意圖

Problem：Clustered Steiner Tree problem (CluSteiner)
Instance：A metric graph $G = (V, E, c)$, and a partition $\mathcal{R} = \{R_i | 1 \le i \le k\}$ of R.
Goal：Find a minimum-cost clustered Steiner tree for $\mathcal{R}$.

我們對題目給出圖例(圖 1.1)，假設給一 metric graph G，其中 required vertex set $R_1 = \{1, 2, 3, 4\}$、$R_2 = \{6, 7, 8, 9\}$、$R_3 = \{11, 12, 13, 14\}$，出來結果可能會如圖所示，黑點為 required vertex，白點為 Steiner 點，圖上所有邊的 cost 總和是最小，而圖中由虛線圈起來的樹就是 *local tree*。

我們接著定義，(u, v)表示點u與點 v 之間的邊，而c(u, v)表示這個邊上的 cost。

Definition 3. 若有一個 edge (u, v)稱作 *Intra edge* 表示u, $v \in T_a$；若有一 edge(u, v)稱作 *Inter edge* 表示$u \in T_a$, $v \in T_b$, $a \neq b$。

Definition 4. *Intra cost* 為 intra edge 上的 cost；*Inter cost* 為 inter edge 上的 cost。

假如有一個圖G'，我們對一個邊$(u,v) \in G'$做 *contraction*，表示用一個新的點 w 來取代點 u 和點 v，而所有與點 u 和點 v 相連的點，全部連到新點 w，我們用$G'[(u, v)]$來表示對G'的邊(u, v)做 contraction。

Triple z 為點集合，且$|z| = 3$，也就是$z = \{u, y, w\}$，假如我們對圖G'上的 triple 做 contraction，我們表示為$G'[z] = G'[(u, y)][(y, w)]$，也就是做了兩次 contraction。$G'[R_1, R_2 \ldots, R_k]$表示對圖$G'$裡的點集合$R_1$到點集合$R_k$分別地 contract。

Definition 5. *Intra triple* 為三個點的集合，且這三個點皆屬於同一個群裡，也就是說$z = \{u, y, w\}$，$u, y, w \in T_a$；*Inter triple* 的三個點皆屬於不同的群裡，也就是$z = \{u, y, w\}$，$u \in T_a$, $u \in T_b$, $w \in T_c$, $a \neq b \neq c$

3. 演算法

在這一節我們詳細敘述了三個 heuristic algorithms 以及會使用到的演算法。演算法裡用到的 mst(U)，U 為點集合，表示對點集合 U 做 MST 演算法求出的值。

Heuristic Algorithm1 Intra_First_algorithm

Input: metric graph G(V, E, c)、
partition $\mathcal{R} = \{R_1, R_2, \ldots, R_k\}$
Output: Clustered Sterner tree $\mathcal{T}$
(1)Call Intra-triple function;
(2)Call Inter-triple function;
(3)Construct k Trees for $\mathcal{R}' = \{R'_1, \ldots, R'_k\}$ by MST algorithm;
for j from 1 to k **do**
$H \cup contract(R'_j)$;
end for
Construct a Clustered Steiner Tree $\mathcal{T}$ for H by MST algorithm;

此演算法的輸入(input)為 metric graph G 還有一個 partition $\mathcal{R}$，輸出(output)為 heuristic algorithms 做出來的 Clustered Sterner tree $\mathcal{T}$。這演算法主要概念分三部分，第一部分會先找出能夠改善 local tree 內部花費(intra cost)的 Steiner vertex，並將該 Steiner vertex 放在改善的 require vertex set 裡面，產生新的 require vertex set R'_j；第二部分是在剩餘的 Steiner vertices 裡找到能夠改善 local trees 之間的花費(inter cost)，並將能改善總 inter cost 的 Steiner vertices 放進點集合H；第三部分為使用前兩部分找出能改善 cost 的 Steiner vertices 建造 Clustered Steiner Tree $\mathcal{T}$。

程式的(1)為上述第一部分，這副程式做法會在後面詳細介紹；程式的(2)是主要第二部分，這副程式一樣會在後面詳細介紹；程式的(3)是先把(1)輸出的$\mathcal{R}'$用 MST 演算法建出 k 個 trees，之後把這 k 個 tree 做 contraction，可以看作把整個 tree 縮成一個點，再把這 k 個點加入H，之後用 MST 演算法對H建造 Clustered Steiner Tree $\mathcal{T}$。

Algorithm 1 Intra-triple

Input: metric graph G(V, E, c)、
partition $\mathcal{R} = \{R_1, R_2, \ldots, R_k\}$
Output: new partition $\mathcal{R}'$

(1)$\mathcal{R}' \leftarrow \emptyset$;
for j from 1 to k **do**
$W_j = \emptyset$, j from 1 to k
end for
(2)**for** every $z = \{u, y, w\}$, $z \subset R_j$,
j from 1 to k **do**
Find Steiner vertex v which minimizes
$\sum_{s\in z} c(v, s)$;
$s\_v(z) = v$;
$c(z) = \sum_{s\in z} c(v, s)$;
end for
(3)**while** there is a z can make win > 0 **do**
(i)find z which $s\_v(z) \neq \emptyset$, and maximizes
$win = mst(R_j) - mst(R_j[z]) - c(z)$;
(ii)**if** $win \leq 0$ **then** continue;
end if
else insert $(R_j, s\_v(z))$;
$mst(R_j) \leftarrow mst(R_j[z])$;
go to step 2;
end else
end while;
(4)**for** j from 1 to k **do**
$R'_j = R_j$;
$\mathcal{R}' \cup R'_j$;
end for

程式的(1)主要是在做初始化動作，W是一個點集合，用來放找到能改善 intra cost 的 Steiner vertex，$\mathcal{R}'$是一個的 partition，用來放新的 required vertex set R'_j。程式的(2)是要對每一個R_j裡的 triple 做紀錄動作，$s_v(z)$記錄哪個 Steiner vertex 相連產生最少 cost，$c(z)$記錄相連產生的最少 cost。程式的(3)是一個無限迴圈，主要做下列兩件事：(i)是找一個 triple，與一個 Steiner vertex 能夠使總 intra cost 省最多，判斷總 intra cost 等式如下

$$win = mst(R_j) - mst(R_j[z]) - c(z)$$

$mst(R_j)$是對該 cluster 用 MST 演算法算出的值，$mst(R_j[z])$是對該 cluster 裡的 triple z 先做 contraction 在用 MST 算出值；(ii)如果沒有能省的 triple，繼續迴圈，直到全部 triples 都檢查完，若有最省，把與之配對的 Steiner vertex 放到R_j裡，並且用$mst(R_j[z])$當成下一個$mst(R_j)$，最後回到程式的(2)繼續重新找新的 triples。程式的(4)是產生$\mathcal{R}'$並輸出。

Algorithm 2 Inter-triple

Input: metric graph G(V, E, c)、
partition $\mathcal{R} = \{R_1, R_2, \dots, R_k\}$
Output: vertex set H
(1)$H \leftarrow \emptyset$, $Y \leftarrow \emptyset$;
(2)**for** R_i, i from 1 to k **do**
Y ∪ contract(R_i);
end for
Y ∪ contract(H);
for every $z = \{u, y, w\}$; $z \subset Y$ **do**
Find Steiner vertex v which minimizes
$\sum_{s\in z} c(v, s), v \notin H$;
$s\_v(z) = v$;
$c(z) = \sum_{s\in z} c(v, s)$;
end for
(3)**while** there is a z can make win > 0 **do**
(i)find z which $v(z) \neq \emptyset$,
and maximizes $win = mst(\mathcal{R}[R_1, \dots, R_k]) - mst(\mathcal{R}[R_1, \dots, R_k, z]) - c(z)$;
(ii)**if** $win \leq 0$ **then** continue
end if
else $insert\ \left(H, s\_v(z)\right)$;
$mst(\mathcal{R}[R_1, \dots, R_k]) \leftarrow mst(\mathcal{R}[R_1, \dots, R_k, z])$;
go to step 2;
end else
end while

程式的(1)是初始化動作，H是一個點集合，用來放能改善 intra cost 的 Steiner vertex，Y 也是一個點集合，放對required vertex set R_i做 contraction 後新的點，如果從 Y 裡選三個點，這三點都屬於不同群。程式的(2) 是要先對每個 required vertex sets 以及 H 做 contraction 後放在 Y 集合裡，之後對這點集合的所有 triple 做記錄動作，$s_v(z)$是記錄 Steiner vertex，看哪個 Steiner vertex 相連會產生最少 cost，$c(z)$記錄相連後產生最少的 cost。程式的(3) 是一個無限迴圈，主要做下列兩件事：(i)是找一個 triple，與一個 Steiner vertex 能夠使總 intel cost 省最多，下列等式為判斷總 intel cost 等式:

$$win = mst(R[R_1, \dots, R_k]) - mst(R[R_1, \dots, R_k, z]) - c(z)$$

$mst(R[R_1, \dots, R_k])$為對 partition R裡的所有R_j做 contraction，可以想像成每個 cluster 裡的 vertex 皆縮成一點，再用 MST 演算法算出值，$mst(R[R_1, \dots, R_k, z])$為把所有 required vertex 皆縮成一點後，在對 triple z 所選到的三個 cluster 在做 contraction；(ii)如果沒有能省的 triple，繼續迴圈，直到全部 triples 都檢查完，若有最省，把與之配對的 Steiner vertex 放到H裡，接著用$mst(\mathcal{R}[R_1, \dots, R_k, z])$當下一個$mst(\mathcal{R}[R_1, \dots, R_k])$，再回到程式的(2)繼續重新找新的 triples。

Heuristic Algorithm 2 Inter_First_algorithm

Input: metric graph G(V, E, c)、
partition $\mathcal{R} = \{R_1, R_2, \dots, R_k\}$
Output: Clustered Sterner tree $\mathcal{T}$
Call Inter-triple function;
Call Intra-triple function;
Construct k Trees for $\mathcal{R}' = \{R'_1, R'_2, \dots, R'_k\}$ by MST algorithm;
for j from 1 to k **do**
$H \cup contract(R'_j)$;

end for
Construct a Clustered Steiner Tree $\mathcal{T}$ for H by MST algorithm;

第二個 Heuristic Algorithm 與上一個 Heuristic Algorithm 很像，差別在於此演算法是先做 Inter-triple function，在做 Intra-triple function，意思就是先行考慮能改善總 inter cost 的 Steiner vertex，在從剩下的 Steiner vertex 去挑選能改善總 intra cost 的 Steiner vertex。還會設計這演算法原因是因為，假如有些 Steiner vertices 能同時大大改善總 intra cost 與總 inter cost 的話，這些 Steiner vertex 便要取捨用來改善哪個 cost，因此兩個方面都要設計演算法來看分配給哪 cost 更好。

Heuristic Algorithm 3 Mixed_algorithm

Input: metric graph G(V, E, c)、
partition $\mathcal{R} = \{\mathrm{R}_1, \mathrm{R}_2, \dots, \mathrm{R}_\mathrm{k}\}$
Output: Construct a Clustered Steiner Tree $\mathcal{T}$
(1)$\mathcal{R}' \leftarrow \emptyset$; $H \leftarrow \emptyset$; $\mathrm{Y} \leftarrow \emptyset$；
for j from 1 to k **do**
$W_j = \emptyset$
end for
(2)**for** every $z = \{u, y, w\}$, $z \subset R_j$, j from 1 to k **do**
Find Steiner vertex v which minimizes $\sum_{s\in z} c(v, s), v \notin H$;
$s\_v(z) = v$;
$\mathrm{c}(z) = \sum_{s\in z} c(v, s)$;
$\mathrm{class(z)} = \mathrm{intra}$;
end for
(3)**for** R_i, i from 1 to k **do**
Y ∪ contract(R_i);
end for
Y ∪ contract(H);
for every $z = \{u, y, w\}$; $z \subset \mathrm{Y}$ **do**
Find Steiner vertex v which minimizes $\sum_{s\in z} c(v, s), v \notin H$;
$s\_v(z) = v$
$\mathrm{c}(z) = \sum_{s\in z} c(v, s)$;
$\mathrm{class(z)} = \mathrm{inter}$;
end for
(4)**while** there is a z can make win > 0 **do**
find z which $s\_v(z) \neq \emptyset$, and maximizes
if class(z) == intra **then**
$win = mst(R_j) - mst(R_j[z]) - c(z)$;
end if
if class(z) == inter **then**
$win = mst(\mathcal{R}[R_1, \dots, R_k]) - mst(\mathcal{R}[R_1, \dots, R_k, z]) - c(z)$;
end if
if $win \leq 0$ **then** continue
end if
else
if class(z) == intra **then**
insert (R_j, s_v(z));
$mst(R_j) \leftarrow mst(R_j[z])$;
go to step 2;
end if
if class(z) == inter **then**
insert (H, s_v(z));
$mst(\mathcal{R}[R_1, \dots, R_k]) \leftarrow mst(\mathcal{R}[R_1, \dots, R_k, z])$;
go to step 2;
end if
end else
end while
(5)**for** j from 1 to k **do**
$R'_j = R_j$;
$\mathcal{R}' \cup R'_j$;
end for
Construct k Trees for $\mathcal{R}' = \{\mathrm{R}'_1, \dots, \mathrm{R}'_\mathrm{k}\}$ by MST algorithm;
for j from 1 to k **do**
$H \cup contract(\mathrm{R}'_\mathrm{j})$;
end for
Construct a Clustered Steiner Tree $\mathcal{T}$ for H by MST algorithm;

第三個 Heuristic Algorithm，這演算法是把能同時改善總 intra cost 與總 inter cost 的 Steiner vertices 依能改善最多 cost 便放到所屬的位置裡，與前兩個演算法不同的是，Steiner vertices 並沒有分要先分到哪位置，而是依哪個位置能改善最多，便分配過去，我們希望研究不同的分配規則看哪個分配規則能普遍最好。

程式的(1)是初始化動作；程式的(2)與(3)是要對每一個 triple 做記錄動作，哪個 Steiner vertex 相連產生最少的 cost，並且記錄類別，class(z)是記錄屬於哪個類別，intra表示這 triple 屬於intra，而inter表示這 triple 屬於inter；程式的(4)會依屬於的類別用不同計算方式來找哪個能改善最多，就把 Steiner vertex 分過去；程式的(5)是在做最後動作，把 Steiner vertex 分完的結果，使用 MST 演算法建出 Construct a Clustered Steiner Tree $\mathcal{T}$。

上面介紹三個演算法的時間複雜度為 $\mathrm{O}(V^5 \log V \times |S|)$，因為我們會先找 triples，這時間複雜度為$\mathrm{O}(V^3)$，接著會對每個 triple 去做 MST 演算法，我們是使用 Kruskal's algorithm，時間複雜度為$\mathrm{O}(\mathrm{E} \log E)$而因為圖是 complete graph，所以 E 為 V^2代入可得$\mathrm{O}(\mathrm{V}^2 \log V)$，在我們的演算法裡，每次選取一個 Steiner vertex 後，以上步驟均要再重做一次，所以還需要乘上 Steiner vertex 數量$|S|$，最後可得$\mathrm{O}(V^5 \log V \times |S|)$.

4. 實驗結果

這一節要給出我們演算法與其他演算法比較的結果，我們使用的資料皆為人造資料，在同一個表格裡，每個值皆為對十個條件相同的人造資料做

出來結果取平均，每一個表格的人造資料所造的條件皆不相同。

表格裡的|G|代表圖的總點數；$|\mathcal{R}|$為 required vertex set R_i的個數，表示總共分幾個群；$|R_i|$表示每一群裡有幾個 required vertices，我們所給的是每群點數皆相同的結果，也就是說 Steiner vertex 的個數便是$|G| - |\mathcal{R}| \times |R_i|$的結果，我們用|S|表示 Steiner vertex 的個數。

表裡面的演算法 MST of MST 為論文[5]裡證明的三倍近似演算法，這演算法的作法為：先對每一個 required vertex set R_i做 MST 演算法，再找每一群之間最短的邊的 cost，且不產生 cycle，把每群 MST 做出來的值與每一群之間最短的邊的 cost 相加，便是這演算法的結果。演算法 Intra First 為我們提出的第一個 heuristic algorithm，先讓 Steiner vertex 根據能省最多的 intra triple 去分配改善 intra cost 後再來改善 inter cost。演算法 inter First 為我們提出的第二個 heuristic algorithm，與第一個 heuristic algorithm 相反，先用 inter triple 去考慮。演算法 Mixed 為考慮全部 intra triple 和 inter triple 能省下的 cost 去分配 Steiner vertex。

由表 4-1 到 4-8，我們可以看出 MST of MST 演算法雖然時間很短，但是我們提出的三個啟發示演算法給的解皆比 MST of MST 好，而我們可以看到 Cost 方面 Intra First < Mixed < Inter First。我們推測如下，我們 Steiner vertex 分配的依據是對我們所選取的三個點去算 cost，能省下最多就分配過去，當我們使用 Intra First 與 Mixed 演算法分配的時候，可能除了所選的三個點還能對其他點省下 cost，所以得到較好的結果，而 Inter First 得到值比較差，有可能是因為分配到 inter triple 的 Steiner 點能在 intra triple 裡能有省更多的 cost，但卻被分到 inter triple，導致總體 cost 較沒那麼好。而 Intra First 會比 Mixed 好一點，我們推測是因為 Steiner vertex 分配到 inter triple 後影響 cost 的群數比分配到 intra triple 後影響的點數更少，導致 Intra First 比 Mixed 好。

在時間方面我們可以看到 Inter First < Intra First < Mixed，在這三個演算法裡，花時間的部分在於找 triple z 與做 MST 運算，每次有 Steiner vertex 被分配出去，triple z 每次都要重新去找，所以 Mixed 是三個演算法裡找 triple 次數裡最多的，不管 Steiner vertex 被分配到 intra triple 或 inter triple 後，都要找新的 intra triple 和 inter triple，相比之下 Intra First 與 Inter First 都只要重新找被分配到相對應的 triple。Inter First 時間會比 Intra First 還要短，我們推測是因為在找改善總 inter cost 的時間比較短，快速找到後，找改善 intra cost 的 Steiner vertex 已經少了一些，於是速度會比較快。

接下來我們要比較各參數影響時間的多寡，我們先看 Steiner vertex 個數的影響，比較表 4-1、4-2 與 4-3 還有 4-6 與 4-9，表 4-2 的 Steiner vertex 個數為表 4-1 的 2 倍，時間約為 2 倍；表 4-3 的 Steiner vertex 個數為表 4-2 的 2 倍，時間約為 3 倍；再來看其他參數大一點的時候，表 4-9 的 Steiner vertex 個數為表 4-6 的 2 倍，時間則約為 2 倍，可以發現，沒有影響到指數倍這麼多。

再來比較表 4-2 與 4-5 還有 4-6 與 4-8，可以發現，當$|R_i|$增加的時候，時間明顯提升了，表 4-5 的$|R_i|$為 4-2 的 2 倍，時間卻為約 17 倍；表 4-8 的$|R_i|$為 4-6 的 1.5 倍，時間約為 5 倍。接著我們看群數$|\mathcal{R}|$對時間的影響，比較表 4-1 與 4-4 還有 4-6 與 4-8，表 4-4 的$|\mathcal{R}|$為 4-1 的 1.5 倍，時間為約 5 倍；表 4-8 的$|\mathcal{R}|$為 4-6 的 2 倍，時間約為 5 倍。

時間花費會因為各參數不同而有所影響，但我們可以看到，影響並沒有到指數倍這麼多，所以我們的演算法確實能在短時間之內給出不錯的答案。

表 4-1. 各演算法比較一

$\|G\|$=70 $\|\mathcal{R}\|$=4 $\|R_i\|$=5 $\|S\|$=50		
Algorithm	**Cost**	**Time[s]**
MST of MST	3578.9	0.0022
Intra First	3491.2	0.0128
Inter First	3505.3	0.0123
Mixed	3493.9	0.0195

表 4-2. 各演算法比較二

$\|G\|$=120 $\|\mathcal{R}\|$=4 $\|R_i\|$=5 $\|S\|$=100		
Algorithm	**Cost**	**Time[s]**
MST of MST	3582.4	0.0028
Intra First	3476.2	0.0294
Inter First	3487.0	0.0249
Mixed	3486.8	0.0408

表 4-3. 各演算法比較三

$\|G\|$=220 $\|\mathcal{R}\|$=4 $\|R_i\|$=5 $\|S\|$=200		
Algorithm	**Cost**	**Time[s]**
MST of MST	3892.3	0.0054
Intra First	3793.5	0.0895
Inter First	3794.5	0.0858
Mixed	3793.1	0.1348

表 4-4. 各演算法比較四

$\|G\|$=130 $\|\mathcal{R}\|$=6 $\|R_i\|$=5 $\|S\|$=100		
Algorithm	**Cost**	**Time[s]**
MST of MST	5315.3	0.0040
Intra First	5140.5	0.0636
Inter First	5142.7	0.0545
Mixed	5142.8	0.1082

表 4-5. 各演算法比較五

$\|G\|=140$ $\|\mathcal{R}\|=4$ $\|R_i\|=10$ $\|S\|=100$		
Algorithm	**Cost**	**Time[s]**
MST of MST	6736.1	0.0029
Intra First	6607.3	0.4911
Inter First	6612.0	0.4318
Mixed	6609.2	0.5671

表 4-6. 各演算法比較六

$\|G\|=150$ $\|\mathcal{R}\|=5$ $\|R_i\|=20$ $\|S\|=50$		
Algorithm	**Cost**	**Time[s]**
MST of MST	15947.3	0.0045
Intra First	15753.8	7.7923
Inter First	15779.5	6.7280
Mixed	15768.1	8.3539

表 4-7. 各演算法比較七

$\|G\|=200$ $\|\mathcal{R}\|=5$ $\|R_i\|=30$ $\|S\|=50$		
Algorithm	**Cost**	**Time[s]**
MST of MST	22977.8	0.0063
Intra First	22804.2	48.4818
Inter First	22833.1	44.6723
Mixed	22810.3	55.9430

表 4-8. 各演算法比較八

$\|G\|=250$ $\|\mathcal{R}\|=10$ $\|R_i\|=20$ $\|S\|=50$		
Algorithm	**Cost**	**Time[s]**
MST of MST	32630.6	0.0127
Intra First	32281.5	38.9017
Inter First	32326.1	32.3780
Mixed	32317.6	39.5898

表 4-9. 各算法比較九

$\|G\|=200$ $\|\mathcal{R}\|=5$ $\|R_i\|=20$ $\|S\|=100$		
Algorithm	**Cost**	**Time[s]**
MST of MST	16261.9	0.0066
Intra First	15995.1	14.4875
Inter First	16022.1	13.8246
Mixed	15999.6	16.5298

5. 結論與未來研究方向

從實驗結果可以看出，我們給的三個 heuristic algorithms 都能在非常短的時間內給出比三倍近似演算法還要好的答案。

我們其實也有在研究 branch and bound 的 exact algorithm，但是現在的 bound 還沒有很好，程式在 Steiner vertex 數量很少時還是需要很長的時間來解，所以在比較時沒有給 exact solution，所以我們未來展望，是希望能夠研究出一個比較好的 bound 來實作我們的 exact algorithm。

我們還對程式效率不滿意，我們希望能再改進我們的程式，還希望以後實驗結果不只有人造資料可以比較，可以找一些實際資料來比較結果。

參考文獻

[1]Du, D. Z., Zhang, Y., & Feng, Q. (1991, October). On better heuristic for Euclidean Steiner minimum trees. In *FOCS (Vol. 91, pp. 431-439).*

[2]Du, D. Z. (1995). On component-size bounded Steiner trees. *Discrete Applied Mathematics, 60(1), 131-140.*

[3] Stanojević, M., & Vujošević, M. (2006). An Exact Algorithm for Steiner Tree Problem on Graphs. *International Journal of Computers, Communications & Control, 11(11).*

[4]Wu, B. Y. (2013). On the Clustered Steiner Tree Problem. In *Combinatorial Optimization and Applications (pp. 60-71)*. Springer International Publishing.

[5] Wu, B. Y, Lin , C. W. & Chen, L. H. (2014)The Steiner ratio of the clustered Steiner tree problem is three. *Unpublished manuscript,* Department of Computer Science and Information Engineering National Chung Cheng University, ChiaYi, Taiwan 621,R.O.C.

[6] Wu, B. Y., & Chao, K. M. (2004). *Spanning trees and optimization problems*. CRC Press.

[7]Zelikovsky, A. Z. (1993). An 11/6-approximation algorithm for the network Steiner problem. *Algorithmica, 9(5), 463-470*.