**計算智慧於工程上的應用**

**SA/TS Homework**

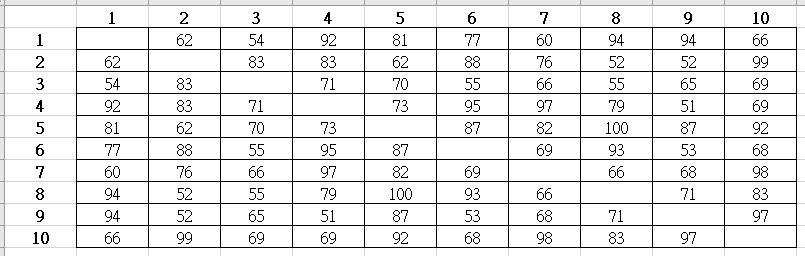
**M11205314**

**張原嘉**

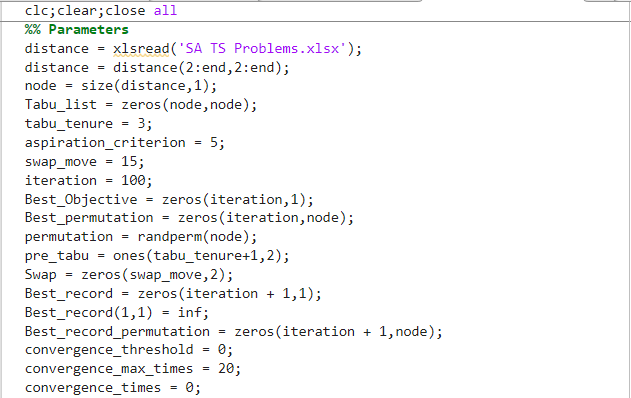
* Programming Language : Matlab



**Q1**



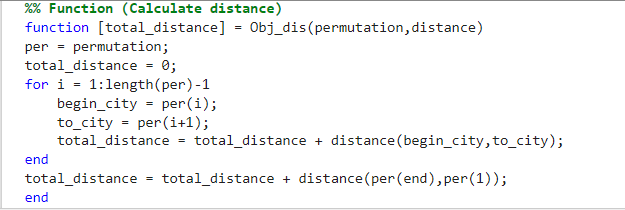
* Algorithm : Tabu Search (TS)
* Code explanations

****

* distance : 代表給定的excel檔案，代表每個地點之間的距離。
* node : 代表地點的數目，10個地點。
* tabu tenure : 設置為3，代表被限制的點會有長達三次的次數不能再被選中。
* aspiration criterion: 設置為5 當被限制的點得到的Objective小於現有的

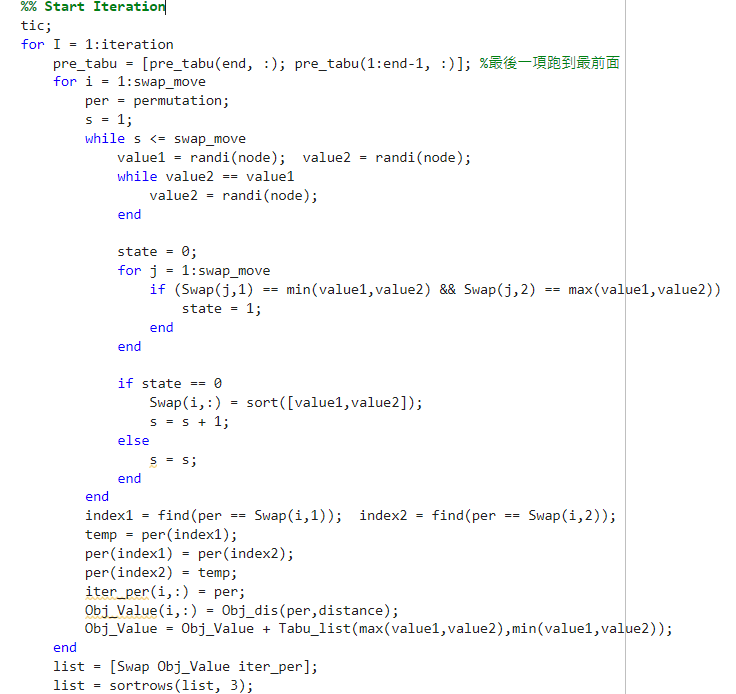
Objective超過5時(因為是最小化問題)，即使在限制中，仍然可以選擇。

* swap\_move : 設置為15，代表每次迭代選取的的swap數目，一次選兩個數字做交換。
* Iteration : 迭代次數設為100次。
* Best\_Objective和Best\_permutation : 用來表示每次迭代的最佳解和對應的排列組合。
* permutation : 初始的排列組合，隨機生成一串1到10隨機排列的組合。
* pre\_tabu : 用來裝曾經選取過的點，先後順序由上面排到下面，short-term memory會根據這個矩陣來分別設置tabu tenure。
* Swap : 用來裝每次選取的swap。
* Best\_record和Best\_record\_permutation : 用來裝每次迭代後的最佳解紀錄和對應的排列組合。
* Convergence\_threshold : 用來偵測收斂用，若最佳解的紀錄變化等於0時，就增加一次收斂的次數。
* Convergence\_max\_time : 最大的收斂次數，當收斂次數達到這個最大值後，代表最佳解紀錄已經沒甚麼變動，達到收斂，即刻停止迭代。
* Convergence\_times : 初始收斂次數，設為0。



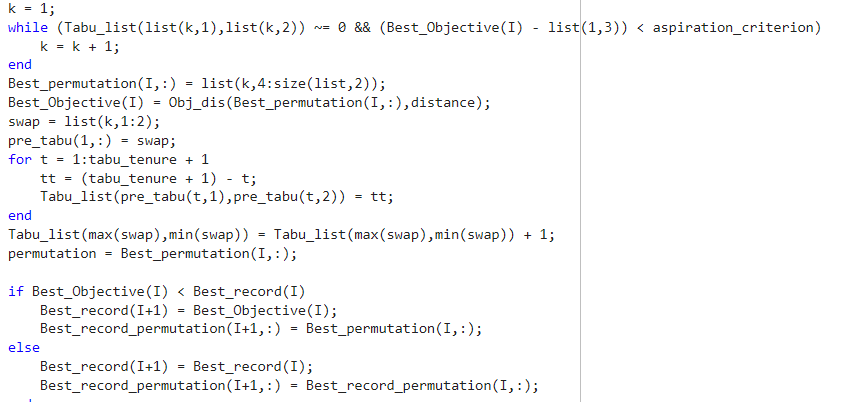
Function : Obj\_dis 用來計算排列組合中，全部行程跑一次總共的距離，最後要回到原點所以會再加個從終點到原點的距離。

* Frequency penalty : short-term memory、long-term memory



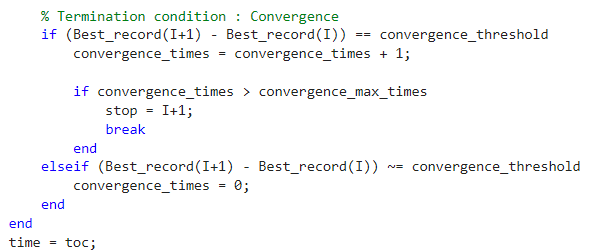
一開始隨機生成swap move，為了避免生成兩個一樣的數字，當value1 = value2的時候，會重新生成value2，除了避免每次swap生成兩個一樣的數字，同時也希望每一列的swap move不要重複生成一樣的組合，比如第一列是[5,7]，第二列也是[5,7]，所以會去偵測每一列swap move確保沒有生成一樣的，若有重複的

state = 0，反之state = 1，接著把選好的每個swap move裡的兩個數字做交換得到新的permutation並用前面寫好的function計算Objective Value，這裡會應用到long-term memory，根據過去選到的次數來對現有的swap做懲罰，所以Objective value會再加上過去被選到的次數(因為是最小化問題，所以相加為懲罰)，也就是Tabu list矩陣的下三角部分，接著把Swap(兩個數字)、Objective Value(值)和對應的permutation(10個數字的排列組合)組合一個list矩陣，並且根據中間的Objective Value由上而下，由小排到大(因為是最小化問題)

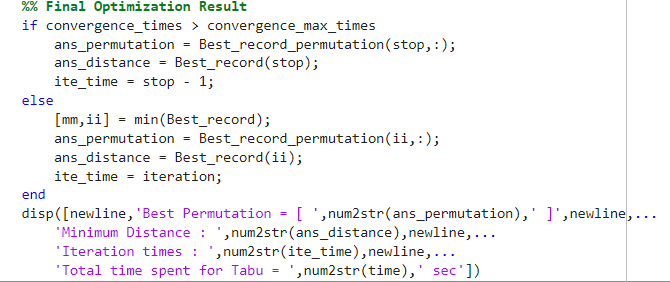


接著是選取的問題，這裡會應用到short-term memory，當第一個swap在被限制中(即代表上三角矩陣含有值)，則會跳過第一個，往下選取，以此類推，但若被限制的swap得到的Objective Value比原來的小5以上(最小化問題)，則可以破除限制直接選取，這裡的5代表一開始設定的aspiration criterion，選取好本次迭代的permutation後，計算出Objective Value並把兩者分別放入Best\_permutation和Best\_Objective中，並且在上三角中(short-terms memory)

更新tabu的次數，並且在tabu list的下三角(long-term memory)中增加一次的次數，swap的兩個數字中，較小的數字排前面代表可以放入上三角矩陣，較大的數字排前面代表可以放入下三角矩陣，接著更新Best\_record和Best\_record\_permutation，這裡如果得到的Objective value比以前小(最小化問題)，就取代掉以前的Best\_record，因為是最小化問題，所以前面的Best\_record初始值設為無限(inf)，代表第一次一定會取代。

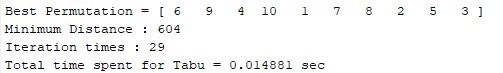


接著是停止條件，若收斂達到停止條件，則提前停止迭代，這裡用的停止條件是當Best\_record前後相差等於0(沒變化)的次數達到20次即停止迭代，下面的elseif代表若Best\_record的值有產生變化，則次數重新計算。

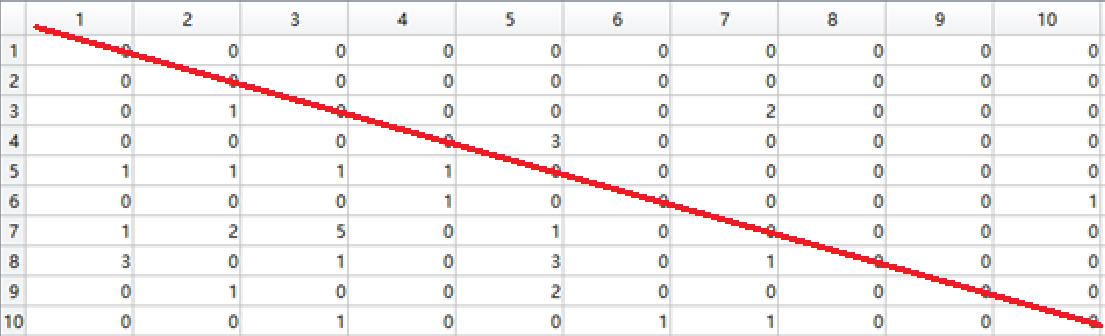


將得到的最佳解(最短距離)與排列組合(Permutation)顯示出來

* Optimization result



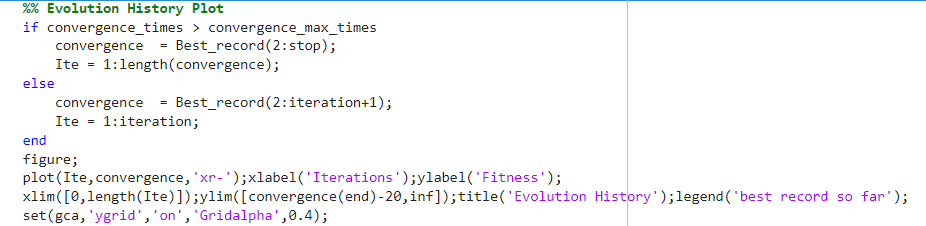
與最終的tabu list

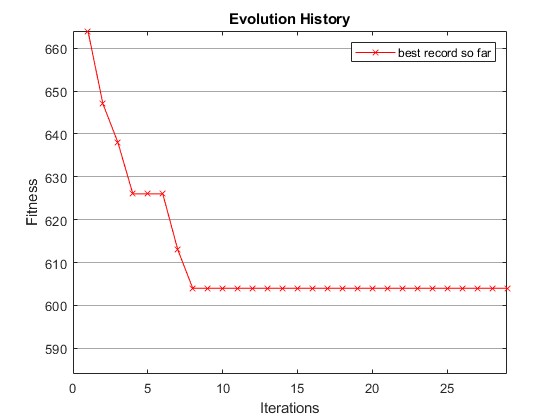


* How many times was the objective function evaluated to find the optimal solution?

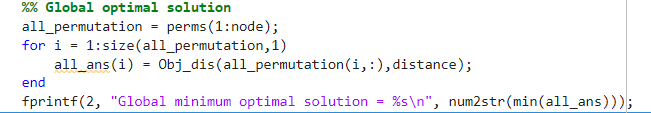
Iteration\*swap move = 29 \* 15 = 435 times

* Convergence History





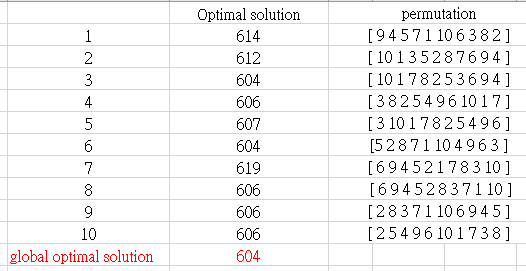
* Use exhaustive enumeration to find out the global optimal solution





這裡使用了matlab函數perms，perms會列出input裡面所有的排列組合，這裡的input是1到10的數字排列，接著把所有排列拿去計算distance，並使用min函數找到最小值，即為global optimal solution，這裡找到的答案是604

* How close is the result compared to the global optimal solution



這裡執行了十次的tabu search所得出的解與global optimal solution來做比較，答案非常接近global optimal solution。

* Matlab Code
* clc;clear;close all
* %% Parameters
* distance = xlsread('SA TS Problems.xlsx');
* distance = distance(2:end,2:end);
* node = size(distance,1);
* Tabu\_list = zeros(node,node);
* tabu\_tenure = 3;
* aspiration\_criterion = 5;
* swap\_move = 15;
* iteration = 200;
* Best\_Objective = zeros(iteration,1);
* Best\_permutation = zeros(iteration,node);
* permutation = randperm(node);
* pre\_tabu = ones(tabu\_tenure+1,2);
* Swap = zeros(swap\_move,2);
* Best\_record = zeros(iteration + 1,1);
* Best\_record(1,1) = inf;
* Best\_record\_permutation = zeros(iteration + 1,node);
* convergence\_threshold = 0;
* convergence\_max\_times = 35;
* convergence\_times = 0;
* %% Start Iteration
* tic;
* for I = 1:iteration
* pre\_tabu = [pre\_tabu(end, :); pre\_tabu(1:end-1, :)]; %最後一項跑到最前面
* for i = 1:swap\_move
* per = permutation;
* s = 1;
* while s <= swap\_move
* value1 = randi(node); value2 = randi(node);
* while value2 == value1
* value2 = randi(node);
* end
* state = 0;
* for j = 1:swap\_move
* if (Swap(j,1) == min(value1,value2) && Swap(j,2) == max(value1,value2))
* state = 1;
* end
* end
* if state == 0
* Swap(i,:) = sort([value1,value2]);
* s = s + 1;
* else
* s = s;
* end
* end
* index1 = find(per == Swap(i,1)); index2 = find(per == Swap(i,2));
* temp = per(index1);
* per(index1) = per(index2);
* per(index2) = temp;
* iter\_per(i,:) = per;
* Obj\_Value(i,:) = Obj\_dis(per,distance);
* Obj\_Value = Obj\_Value + Tabu\_list(max(value1,value2),min(value1,value2));
* end
* list = [Swap Obj\_Value iter\_per];
* list = sortrows(list, 3);
* k = 1;
* while (Tabu\_list(list(k,1),list(k,2)) ~= 0 && (Best\_Objective(I) - list(1,3)) < aspiration\_criterion)
* k = k + 1;
* end
* Best\_permutation(I,:) = list(k,4:size(list,2));
* Best\_Objective(I) = Obj\_dis(Best\_permutation(I,:),distance);
* swap = list(k,1:2);
* pre\_tabu(1,:) = swap;
* for t = 1:tabu\_tenure + 1
* tt = (tabu\_tenure + 1) - t;
* Tabu\_list(pre\_tabu(t,1),pre\_tabu(t,2)) = tt;
* end
* Tabu\_list(max(swap),min(swap)) = Tabu\_list(max(swap),min(swap)) + 1;
* permutation = Best\_permutation(I,:);
* if Best\_Objective(I) < Best\_record(I)
* Best\_record(I+1) = Best\_Objective(I);
* Best\_record\_permutation(I+1,:) = Best\_permutation(I,:);
* else
* Best\_record(I+1) = Best\_record(I);
* Best\_record\_permutation(I+1,:) = Best\_record\_permutation(I,:);
* end
* % Termination condition : Convergence
* if (Best\_record(I+1) - Best\_record(I)) == convergence\_threshold
* convergence\_times = convergence\_times + 1;
* if convergence\_times > convergence\_max\_times
* stop = I+1;
* break
* end
* elseif (Best\_record(I+1) - Best\_record(I)) ~= convergence\_threshold
* convergence\_times = 0;
* end
* end
* time = toc;
* %% Final Optimization Result
* if convergence\_times > convergence\_max\_times
* ans\_permutation = Best\_record\_permutation(stop,:);
* ans\_distance = Best\_record(stop);
* ite\_time = stop - 1;
* else
* [mm,ii] = min(Best\_record);
* ans\_permutation = Best\_record\_permutation(ii,:);
* ans\_distance = Best\_record(ii);
* ite\_time = iteration;
* end
* disp([newline,'Best Permutation = [ ',num2str(ans\_permutation),' ]',newline,...
* 'Minimum Distance : ',num2str(ans\_distance),newline,...
* 'Iteration times : ',num2str(ite\_time),newline,...
* 'Total time spent for Tabu = ',num2str(time),' sec'])
* %% Evolution History Plot
* if convergence\_times > convergence\_max\_times
* convergence = Best\_record(2:stop);
* Ite = 1:length(convergence);
* else
* convergence = Best\_record(2:iteration+1);
* Ite = 1:iteration;
* end
* figure;
* plot(Ite,convergence,'xr-');xlabel('Iterations');ylabel('Fitness');
* xlim([0,length(Ite)]);ylim([convergence(end)-20,inf]);title('Evolution History');legend('best record so far');
* set(gca,'ygrid','on','Gridalpha',0.4);
* %% Global optimal solution
* all\_permutation = perms(1:node);
* for i = 1:size(all\_permutation,1)
* all\_ans(i) = Obj\_dis(all\_permutation(i,:),distance);
* end
* fprintf(2, "Global minimum optimal solution = %s\n", num2str(min(all\_ans)));
* %% Function (Calculate distance)
* function [total\_distance] = Obj\_dis(permutation,distance)
* per = permutation;
* total\_distance = 0;
* for i = 1:length(per)-1
* begin\_city = per(i);
* to\_city = per(i+1);
* total\_distance = total\_distance + distance(begin\_city,to\_city);
* end
* total\_distance = total\_distance + distance(per(end),per(1));
* end